「富岳」の次世代となる新たな
フラッグシップシステムにおける
基本設計技術報告書

第1.1版（公開版）

国立研究開発法人 理化学研究所

令和8年5月29日

概要

文部科学省は次世代の高性能計算基盤の整備に向け、システムや新計算原理との連携、運用技術などの観点で次期フラッグシップスーパーコンピュータを検討するための、次世代計算基盤の調査研究プロジェクトを2022年8月より開始した。「特定先端大型研究施設の共用の促進に関する法律」に基づくスーパーコンピュータ「富岳」（以下、「富岳」という）の運用主体である国立研究開発法人理化学研究所（以下、「理研」という）は、次世代計算基盤の調査研究プロジェクトに参画し、次世代計算基盤としてあるべきシステムや運用の方向性について調査・検討を進めてきたところ、2024年6月に次期フラッグシップスーパーコンピュータの開発・整備プロジェクトを推進するための実施主体として選定された。その後、文部科学省は2030年頃に運用可能なフラッグシップスーパーコンピュータを整備するために、2025年1月より当該開発・整備プロジェクトを開始させた。

理研は、2024年6月の文部科学省HPCI計画推進委員会「次世代計算基盤に関する報告書最終取りまとめ」を受け、次世代計算基盤の調査研究プロジェクトでの検討結果も踏まえながら、「富岳」の次世代となる新たなフラッグシップシステムの開発・整備プロジェクトを進めている。

本システムのコードネームを以下「富岳NEXT」と呼び、その開発・整備プロジェクトを「「富岳NEXT」プロジェクト」と呼ぶ。我が国の科学技術・イノベーションが世界をリードし、社会や産業の発展に繋げるためには、シミュレーションとAIの両者において世界最高水準の性能を達成し、さらにシミュレーションとAIとが密に連携して処理を行いつつ、科学上の仮説生成や実証を含むサイエンスを自動化・高度化する「AI for Science」のためのフラッグシップシステムの実現が欠かせない。今後のAI for Scienceの発展を見据えつつ、既存HPCアプリで現行の5～10倍以上の実効計算性能、AI処理でゼッタ（Zetta）スケールのピーク性能を念頭に50EFLOPS以上の実行性能を実現するシステムを開発・整備し、シミュレーションとAIの融合により総合的に5～10倍の実効計算性能向上分を超える数十倍のアプリ実行高速化を達成することを目標とした。

理化学研究所は公募により富士通とNVIDIAを共同開発先に決定し、富士通とは2025年6月より、NVIDIAとは2025年8月より共に2026年2月まで「富岳NEXT」の基本設計を実施した。

本報告書は3章から構成されている。第一章では開発方針および、基本設計以降の開発スケジュール案を述べる。第二章では「富岳NEXT」のシステム目標性能を述べ、第三章では基本設計の詳細検討結果をまとめている。

目次

1. 開発方針

1.1 「富岳 NEXT」プロジェクト基本設計概要

国立研究開発法人理化学研究所は、2024年6月の文部科学省HPCI計画推進委員会「次世代計算基盤に関する報告書最終取りまとめ」を受け、次世代計算基盤の調査研究プロジェクトでの検討結果も踏まえながら、「富岳」の次世代となる新たなフラッグシップシステムの開発・整備プロジェクト（「富岳NEXT」プロジェクト）を2025年1月より開始し、進めているところである。我が国の科学技術・イノベーションが世界をリードし、社会や産業の発展に繋げるためには、シミュレーションとAIの両者において世界最高水準の性能を達成し、さらにシミュレーションとAIとが密に連携して処理を行いつつ、科学上の仮説生成や実証を含むサイエンスを自動化・高度化する「AI for Science」のためのフラッグシップシステムの実現が欠かせない。今後のAI for Scienceの発展を見据えつつ、既存HPCアプリで現行の5～10倍以上の実効計算性能、AI処理でゼッタ（Zetta）スケールのピーク性能を念頭に50EFLOPS以上の実行性能を実現するシステムを開発・整備し、シミュレーションとAIの融合により総合的に5～10倍の実効計算性能向上分を超える数十倍のアプリ実行高速化を達成することを目標としている。

「富岳NEXT」では、アプリケーションファーストを理念とし、電力制約下でも上記目標を達成するために、「富岳」で培ったアプリケーションソフトウェア等の資産を有効活用できる電力効率の高いCPU部と、帯域重視の演算処理加速部を組み合わせた、高帯域及びヘテロジニアスなノードアーキテクチャを基本構成としたシステムを想定し、本プロジェクトの一環として2025年4月より「富岳NEXT」開発に向けた基本設計を実施してきた。基本設計では理化学研究所が開発主体となりつつ2025年度を通して開発を行うと同時に、公募により2025年6月より富士通会社を、2025年8月より NVIDIAを共同開発先として決定し、両ベンダとは2026年2月まで共同で基本設計を行った。

基本設計で実施した項目は、主として以下となる。ハードウェアについてはCPUやGPUのプロセッシングエレメント、システムハードウェア、ネットワークなどのシステム構成要素の機能・性能定義や利用するデバイス技術の選定等を行った。システムソフトウェアについては、その仕様検討、機能設計を進めるとともにプロトタイプ実装等を行った。また、これらは理化学研究所および共同開発ベンダが協力しつつ、様々なアプリケーションにおいて高い実効性能を得るためにも、「富岳NEXT」の利用者として想定されるアプリケーション開発者とともに協調設計（Co-design）を進めながら実施した。

1.2 「富岳 NEXT」プロジェクトの開発方針

「富岳NEXT」の開発にあたっては、世界最高水準のシステムを実現するだけでなく、科学的成果の最大化、情報産業への世界的な訴求力を意識した国産技術の高度化と技術継承、グローバルマーケットへの展開も念頭に置きつつプロジェクトを推進することが求められている。図 1-1は「富岳NEXT」の開発戦略をまとめたものである。我が国の科学技術・産業競争力を強化することを念頭に、特に、①Made with Japan、②技術革新、③持続性／継続性、の3点を柱に据えることとした。

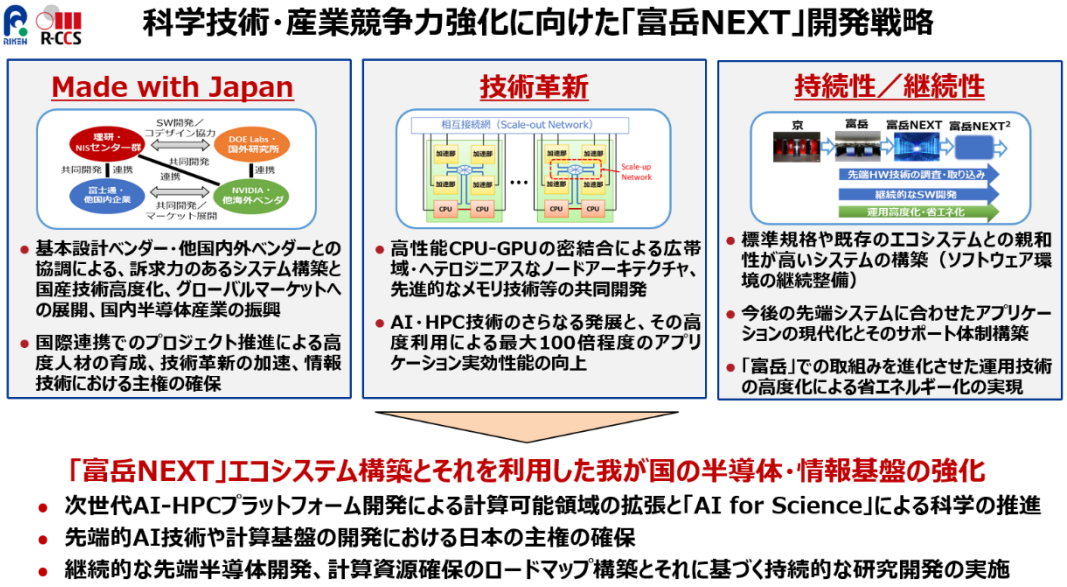


図 1-1 「富岳 NEXT」の開発戦略概要

上記の開発戦略に基づき、費用対効果を最大限にしつつ進めていくため、基本設計では以下の開発方針を設定して業務を実施することとした。

1. CPU部としては、これまで富岳で蓄積されたアプリケーションやシステムソフトウェア資産が活用できるよう、富岳とバイナリレベルで互換性を持つことを原則とする。
2. 加速部としては、プログラミングや性能最適化の観点からユーザに使いやすいものとなるようオープンなソフトウェアエコシステムが整備されている必要がある。また、「富岳NEXT」の稼働前からコード移植を効率的に実施できるよう、加速部アーキテクチャが現状で広く利用可能でなければならない。そこで、大規模なスーパーコンピュータにおいて既に活用実績を持つGPUを演算加速部として導入する
3. 成果物である筐体や冷却技術等のハードウェアシステム自身やその設計資産に関し、それらがエコシステムの一部に組み込まれて世界的に発展していくことも重要であり、そのためには、高コストにならず幅広い需要に適応できるシステムの開発を行う必要がある。そのため、各コンポーネントやシステム設計にあたっては、できる限りオープンな規格を採用し、信頼性レベルも必要十分な範囲となるよう設計を行うべきである。
4. コンパイラやライブラリ等のシステムソフトウェアについては、開発終了後の継続的な機能拡張やコードのメンテナンスの観点から、できる限りオープンソースのソフトウェアを導入する。また、開発したソフトウェアに関しても、ユーザや開発関係者による改変や機能拡張を通じて「富岳NEXT」やコミュニティの発展に資するべく、原則としてオープンソースソフトウェアとして公開する。商用のソフトウェアを導入する場合においても、将来を含めて定期的にアップデートされるものを採用することを原則とする。

5. 「富岳NEXT」プロジェクトにおいて開発するシステム全体や主要なコンポーネント、またソフトウェアについては、高い市場競争力を持つようにコスト・性能に関して十分に検討する。また、整備時期において求められる計算資源の機能についての把握に努めつつ設計を行う。

1.3 「富岳 NEXT」開発スケジュール

図 1-2に現時点で想定する「富岳NEXT」の開発スケジュールを示す。

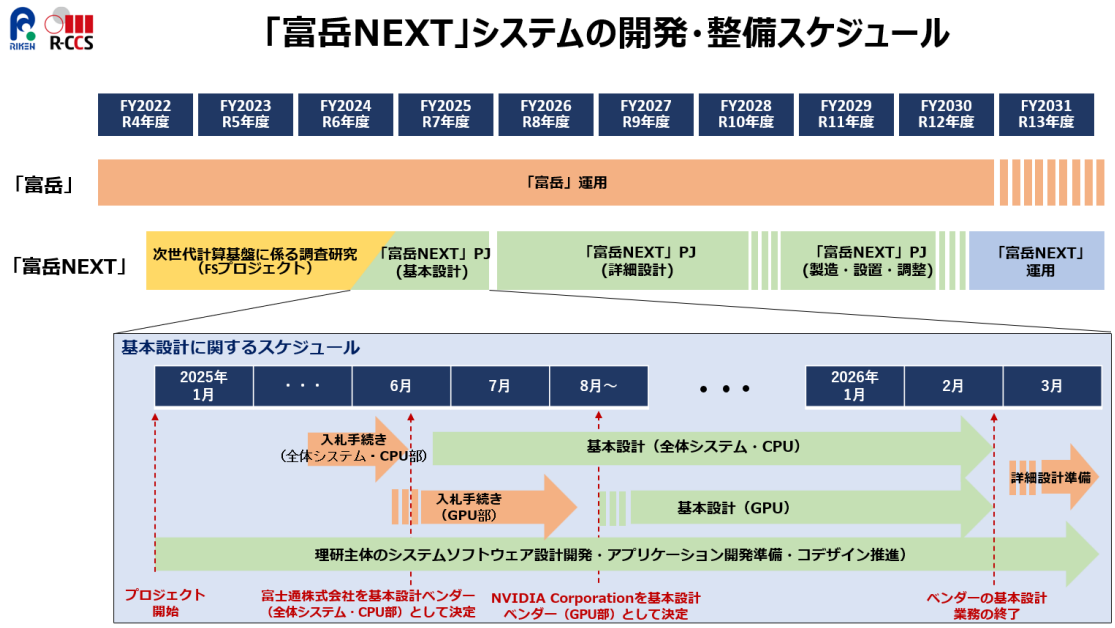


図 1-2 「富岳 NEXT」システム開発・整備スケジュール

2. 目標性能

2.1システム性能目標

表 2- 1 に示すシステム目標性能値を仕様と定め、基本設計を実施した。本性能値は、「次世代計算基盤に関する報告書 最終取りまとめ」に示された次の性能目標項目に基づき、文部科学省の次世代計算基盤に係る調査研究事業でまとめられたものである。40MWのシステム消費電力を上限として、主に従来のHPCアプリケーション向けのFP64ベクトル性能、AI向けの低精度演算性能であるFP16やBF16、FP8の行列演算性能とSparsityを利用する際の性能、メインメモリサイズおよびバンド幅に対し、目標値を与えている。

- 既存 HPC アプリケーションで現行の 5～10 倍以上の実効計算性能
- AI 処理でゼタ（Zetta） FLOPS スケールのピーク性能を念頭に 50EFLOPS 以上の実効性能
- シミュレーションと AI の融合により、総合的に数十倍のアプリケーション高速化を目標

表 2-1 基本設計におけるシステム目標性能値

項目	CPU	加速部	「富岳」	「富岳」比
合計ノード数	3400ノード以上		158,976	
理論 FP64ベクトル性能	48 PFLOPS以上	2.6 EFLOPS以上	537 PFLOPS	x5.7 以上
理論 FP16/BF16行列演算性能	1.5 EFLOPS以上	150 EFLOPS以上	2.15 EFLOPS	x70.5 以上
理論 FP8行列演算性能	3.0 EFLOPS以上	300 EFLOPS以上	—	
Sparsity考慮の 同理論性能	ー	600 EFLOPS以上	—	
メインメモリサイズ	10 PiB以上	10 PiB以上	4.85 PiB	x4.1 以上
メインメモリバンド幅	7 PB/s以上	800 PB/s以上	163 PB/s	x4.9 以上
合計消費電力	計算部を40MW以下で運用		約30 MW	

3. 基本設計検討詳細（システム検討詳細）

本節では、システム基本設計において検討を行った、アーキテクチャ、アプリケーション開発、システムソフトウェア、ストレージとデータ管理、施設・電力・冷却設計、および運用設計の各項目について述べる。アーキテクチャに関しては、検討を行った全体システム、計算ノード、Scale-outネットワークについて説明した後、基本設計に対して設けたコデザイン項目を示す。また、CPUおよびGPUの性能モデル、メモリ性能特性、およびアプリケーションの通信パターンに関する調査について概説する。

アプリケーション開発については、開発体制と計画、アプリケーション開発環境、アプリケーション開発、アプリケーションからのコデザインフィードバック、富岳NEXT時代に向けた新しいアプリケーションの開発、連携について述べる。システムソフトウェアについては、プログラミング環境、数値計算ライブラリ・ミドルウェア、通信ライブラリ、AIソフトウェアについて述べる。ストレージとデータ管理については、アーキテクチャ、ストレージシステム活用支援、性能測定方法とベンチマーク手法、複数予算想定下の構成検討とコスト推定、詳細設計での検討事項について述べる。

3.1アーキテクチャ

3.1.1 全体システム

3.1.1.1 概要

図 3-1に、「富岳NEXT」の全体システム構成の概略を示す。「富岳NEXT」の全体システムは、計算ノードを含む本体システム、全体システムとデータを共有する外部記憶装置である共有ストレージ、及びフロントエンド計算機群を含む周辺装置から構成される。

「富岳NEXT」の本体システムは、以下の特徴を持つ。

- Arm CPU FUJITSU-MONAKA-X を搭載
  - Armv9 SVE2/SME2 サポートによる高い HPC、及び AI 性能
  -
- ヘテロジニアスな計算ノードアーキテクチャ
  - FUJITSU-MONAKA-X と GPU を組み合わせた構成による高い電力効率
- データセンター(DC: Data Center)を含む広範囲ユーザへの商用展開を見据えた計算ノード設計
  - 標準規格のラックに搭載可能な計算ノードシャーシとすることにより、ユーザの既存施設やラック設備への導入を容易化

本体システムは、複数のGPUを用いた並列計算のために GPU 間を接続する Scale-up ネットワーク、複数の計算ノードを用いた並列計算のために計算ノード間を接続する Scale-out ネットワークを有する。また、全体システムは外部記憶装置とデータをやり取りするためのストレージネットワーク、全体システム内の計算ノードや周辺装置の管理・制御に用いるシステム管理系ネットワークを有する。

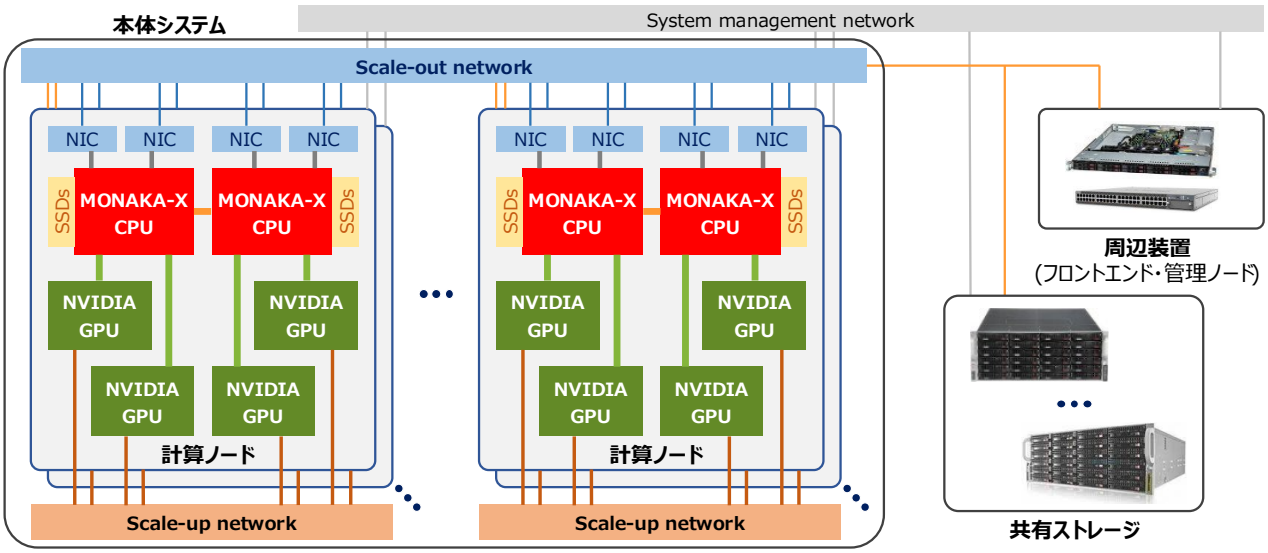


図 3-1 全体システムの概要

3.1.2 計算ノード

3.1.2.1 CPU 部

「富岳NEXT」システムには、富士通社開発のCPU「FUJITSU-MONAKA-X」を搭載する。「FUJITSU-MONAKA-X」CPU は、次世代データセンター向けのCPU「FUJITSU-MONAKA」の技術を継承するとともに、AI 推論向けの機能拡張、メモリ・IO バンド強化を行い、HPC、データセンター、エッジコンピューティング等、幅広い分野への適用を実現する。

「FUJITSU-MONAKA-X」CPU の特徴を、以下に示す。

- 最先端テクノロジ (1.4nm)
- 3D チップレット
- メニーコア
- 低消費電力 (超低電圧)
- 信頼性、セキュリティ
- ベクトルエンジン (SVE2)
- 行列積エンジン (SME2)
- GPU 密結合 (NVLink-C2C)

「FUJITSU-MONAKA-X」CPU は、3Dチップレットを採用し、ダイ間の3 次元接続によるデータ移動の短縮とアンコア関連ダイへの安定テクノロジの採用により、高性能・低消費電力・低コストを同時に実現する。

3.1.2.2 加速部

3.1.2.2.1 基本的な GPU システム構成

富岳NEXT GPUは、モジュラー型マルチチップモジュール（MCM）アーキテクチャを採用し、従来のハードウェアの物理的なスケーリング限界を克服している。タイル型の論理設計を採用することで、ノード全体の高速接続を確保しながら、計算スループットとメモリ容量を大幅に向上させている。このスケーラブルなアプローチにより、多様なワークロード要件に合わせて、ピーク性能と運用効率の柔軟なバランスを実現できる。

3.1.2.2.2 GPU の演算性能

GPUの計算設計における中心的な原則は、ベクトル計算（ストリーミングマルチプロセッサのスループットとスレッドレベルの並列性）とテンソル計算（行列演算のパフォーマンス）の間で最適なバランスを実現することである。この主要な設計上のトレードオフは、各計算の種類にどの程度重点を置くかに応じて、2つの主要な領域におけるシステムの有効性を決定する。

- 高性能スカラーおよびベクトル計算に依存する HPC シミュレーションワークロード
- 集中的なマトリックス演算を中心とする AI ワークロード

3.1.2.2.3 メモリ設計

GPU性能に大きく影響する重要な要素として、基本設計では複数のメモリ構成オプションが提示された。これらのオプションの最終的な選択は、技術の成熟度、製品要件、スケジュール制約などの要素を考慮しながら、詳細設計段階でより詳細に検討される。

初期ベースラインに加えて、いくつかのアーキテクチャ候補が評価されている。最終的な選択は、主に想定されるワ

ークロードの具体的な要件に基づいて行われ、容量、バンド幅、電力効率のバランスが最適化される。

3.1.2.3 加速部 Scale-up 接続

最も効果的なスケールアップ戦略を決定するために、初期設計段階では、2つの確立された相互接続アーキテクチャの比較評価を実施する。将来的には様々なGPU数への検討に拡張される可能性があるが、現在の評価では、NVIDIAで実績のある既存のシステム標準である以下の2つの構成に焦点を当てている。

- NVL72（スイッチベースのスケールアップ）：72 基の GPU が専用 NVLink スイッチのファブリックを介して相互接続された大規模構成。このアプローチは、高密度の計算ノード全体で最大の総合スループットと統合メモリを優先する。
- NVL4（ブリッジベースのスケールアップ）：NVLink ブリッジを使用して 4 つの GPU を相互接続する、よりコンパクトな構成。このモデルは、単一のコンピューティングトレイ内で見られる標準的なノード内接続を表す。

3.1.2.3.1 NVL72

NVL72構成は、NVIDIA現世代のアーキテクチャの高密度進化版であり、大規模な計算向けに最適化されている。これは、現在の世代システム（Grace Blackwell GB200 NVL72）とほぼ同じであるが、全体的な密度が高くなっている。2026年後半以降にリリースが予定されているVera Rubin世代では、各ラックに36個のCPUと72個のGPUが搭載され、中央に9個のスイッチと9個のNVLinkスイッチが配置される予定である。72ウェイの相互接続を可能にするために、NVLinkドメイン内にパッシブ銅線バックプレーンが設置され、その設置用のスペースも割り当てられている。

3.1.2.3.2 NVL4

NVL72構成はスイッチベースの統合に重点を置いているのに対し、NVL4ベースのラックは、アーキテクチャの複雑さをバックプレーンからネットワーク層へと移行するモジュール式の代替手段を提供する。NVL4ラック構成は、Grace BlackwellやVera RubinのNVL72設計における集中型スイッチトレイモデルとは異なる。この構成では、GPUはノードレベルでNVLinkブリッジを介して相互接続されるため、パッシブ銅線バックプレーンと中央のNVLinkスイッチは不要となる。NVLinkスイッチトレイを省くことで、標準Leafスイッチ（L1）をラックに直接統合できる可能性がある。これらのLeafスイッチは、コンパクトなNVLinkスイッチトレイと比較して多くのラックスロットを占有するが、ラック内で統合することで、同一の物理エンクロージャ内で第1レベルのスケールアウト接続が可能になり、システム全体のフットプリントが実質的に向上する。

3.1.3 Scale-out ネットワーク

Scale-outネットワークは、「富岳NEXT」における計算ノード間を接続し、複数の計算ノードを用いた並列計算のための通信を担う。Scale-outネットワークは主に、ネットワークスイッチ、NIC、ケーブル等により構成される。今年度は、多数のGPUを効率的に接続するためのネットワーク・トポロジや、テクノロジーロードマップに基づく想定性能や実現可能性について、検討を行った。

3.1.3.1 富士通想定

富士通が想定しているScale-outネットワーク構成では、4つのレールを有するRail optimized fat-tree (Quad-rail optimized fat-tree)トポロジを採用する。なお、本構成は消費電力やコストを見積もるためのベンダーニュートラルな構成である。

64計算ノードを1グループとし、各グループは4つのTier-1スイッチを有する。グループ内の4つのTier-1スイッチはそれぞれが1つのrailに対応し、各計算ノードに搭載されたScale-out NICのうち同一番号をもつものは同一のTier-1スイッチに接続される。

3.1.3.2 NVIDIA 案

2026年後半に導入予定のNVIDIA社のVera-Rubin世代のテクノロジをベースラインとしてScale-outネットワークの基本設計を検討した。提案されたシステムには以下の新しい技術が組み込まれる。

- Spectrum-X Ethernet: 最適化された適応ルーティングと輻輳制御機能
- Spectrum-X Switch: より高ラディックス、低レイテンシを実現するための次世代スイッチ ASIC
- ConnectX NIC: ノードあたりに要求される高インジェクションバンド幅を実現するための先進ネットワークインターフェース
- NVIDIA 案の Scale-out ネットワークトポロジは、古典的な 2 層 Fat-tree 構成で L1（Leaf）層と L2（Spine）層で構成され、ノンブロッキングかつ高バンド幅のファブリックを確保する。
- L1（Leaf）スイッチはラック内における主要な集約ポイントとして機能する。物理的には、スイッチがコンピューティングラックに配置されていない場合もある。
- L2（Spine）スイッチは、システム全体のグローバル通信のためのラック間接続として機能する。

例として、NVL72ラックあたり4台のLeafスイッチを備えた構成では、最大1:1のオーバーサブスクリプション比をサポートできる。より現実的な4:1のオーバーサブスクリプション比の構成例では、L2スイッチ144、L1スイッチを最大で1152という構成が考えられる。このオーバーサブスクリプション比とラック（またはGPU）の総数から設計に必要なL2スイッチの数が決定される。NVIDIAがこれまでに構築したAIシステムのデフォルト構成では、オーバーサブスクリプション比を1:1とする2層Fat-tree構造が一般的に採用されている。詳細設計フェーズでは、HPC向けの要件を定義し、テーパ比を含めて設計の具体化を行う。

提案されたトポロジでは、NVL72構成のベースラインネットワークアーキテクチャを想定しているが、NVL4による構成にも適用可能であり、Scale-upの構成選択に関わらずScale-outネットワークの性能は一貫して維持される。NVL72とNVL4による構成では、2層のFat-tree構成は同じであるが、ラックあたりのGPU数、L1スイッチの規模や種類、ケーブルの配線は異なるため、詳細設計フェーズでは、エリア、ワーキンググループをまたいだ議論が必要となる。一方で、Scale-outネットワークはScale-upとは独立して設計されるため、NVL72、NVL4などのScale-up構成

に関わらず、一貫した設計が適用できると想定される

NVIDIA案で採用されるSpectrum-X ネットワークスイッチは、ハードウェアベースのパケットレベルの適応ルーティングと、プロアクティブな輻輳制御機能を持つ。これにより、データの再送頻度の低減や、特定のフローに対する優先的な処理等、高性能かつ柔軟な通信制御が可能になる。また、In-band Network Telemetryを活用し、単一のワークロードによるシステム全体の通信性能低下を防ぐことができる。さらに、マルチプレーンアーキテクチャにより、部分的な故障による致命的なジョブの停止を防止する。これにより、耐障害性の高い堅牢なネットワークシステムが構築可能である。

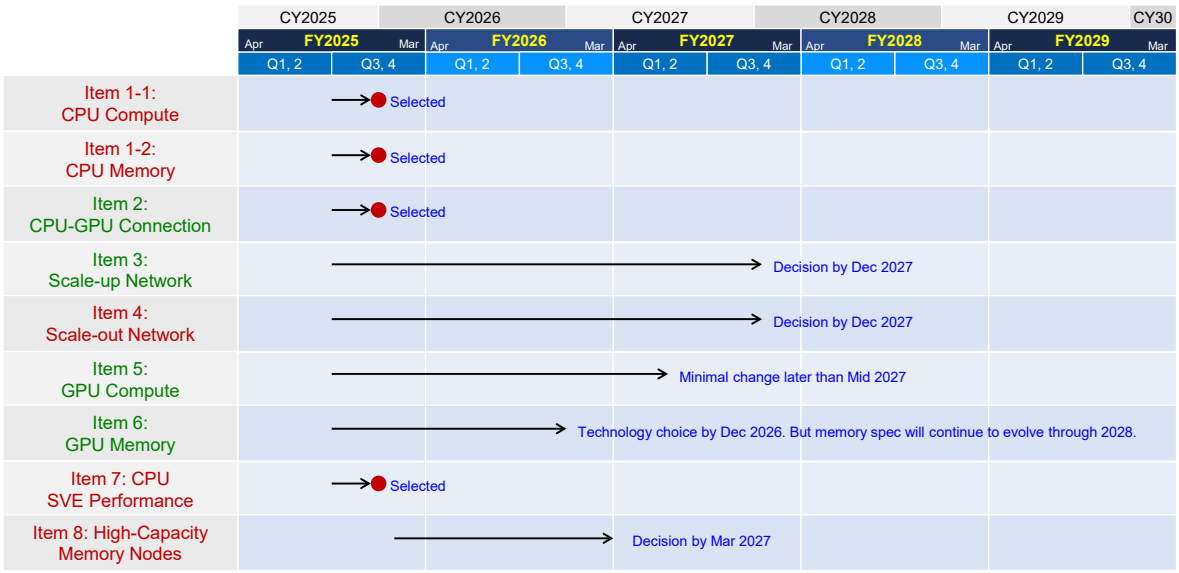
最後に、電力効率の向上、故障率の低減、リンクフラップの削減を目的として、Co-Packaged-Optics (CPO)の導入について検討を行った。システム全体の消費電力を最適化するために、ネットワーク、CPU、GPU、スイッチ等の選定を含む、包括的な検討と設計が必要である。

3.1.4 コデザイン項目

基本設計においては、実現可能性のある技術的選択肢に対し、アプリケーションとのコデザインにより設計空間の探索を行い第一候補となる技術や諸元を選定するために、Item 1からItem 8までの次のアーキテクチャコデザイン項目を設けた。

Item 1: CPU Specification
Item 2: CPU-GPU Connection
Item 3: Scale-up Network
Item 4: Scale-out Network
Item 5: GPU Compute Specification
Item 6: GPU Memory
Item 7: CPU SVE Performance
Item 8: Necessity of High-Capacity Memory Nodes

表 3-5 コデザインの線表



また、Item 1-1, 1-2, 2, 7の項目について、システム基本設計期間中に第一候補の選定がなされた。以下、いくつかの項目の詳細を述べる。

3.1.4.1 Item 3: Scale-up Network

本項目は、Scale-upネットワークに関する設計の技術的選択肢であり、第一候補の選定期限は2027年12月末である。Scale-upドメインサイズ（GPU数）とGPUあたりのバンドに関するいくつかの選択肢について、アプリケーションの性能感度の点で望ましい選択を問うている。Scale-upドメインサイズを増やすことで実際のアプリケーション性能をどの程度向上できるかを評価する必要がある。現時点では、本項目はItem 4のScale-outネットワークとは独立した項目として検討を行う。基本設計においては技術的選択肢がまとめられたのみであり、今後引き続きコデザインを実施する。

3.1.4.2 Item 4: Scale-out Network

本項目は、Scale-outネットワークに関する設計の技術的選択肢であり、第一候補の選定期限は2027年12月末である。Scale-outネットワークの GPUあたりインジェクションバンドに関するいくつかの選択肢について、アプリケーションの性能感度の点で望ましい選択を問うている。ここではネットワーク・トポロジに関する詳細は提示していないが、Fat treeベースの構成を想定することとしている。また、GPUあたりのバンドであり、4 GPU構成のノードあたりのインジェクションバンドはその4倍となる。CPUあたりのバンドはここでは検討しないが、現時点では、4 GPU分のノードあたりバンドを利用できることを想定する。

基本設計においては技術的選択肢がまとめられたのみであり、今後引き続きコデザインを実施する。

3.1.4.3 Item 5: GPU Compute Specification

本項目は、GPUの計算性能に関する設計の技術的選択肢であり、第一候補の選定期限は2027年半ばである。現時点でのGPUソケットあたり計算性能例が提示されており、これに関してアプリケーションの実効性能感度を問うている。その他、今後検討を予定する項目として、ベクトル計算性能とテンソル（行列）計算性能の比、テンソルコアの演算精度のバランス、を想定する。基本設計においては計算性能例が提示されたのみであり、今後引き続きコデザインを実施する。

3.1.4.4 Item 6: GPU Memory

本項目は、GPUのメモリ性能に関する設計の技術的選択肢であり、第一候補の選定期限は2026年末である。容量とバンドの異なるGPUメモリ技術のいくつかの選択肢について、アプリケーション性能の観点で望ましい選択を問うている。

基本設計においては技術的選択肢がまとめられたのみであり、今後引き続きコデザインを実施する。

3.1.4.5 Item 8: Necessity of High-Capacity Memory Nodes

本項目は、標準的な計算ノードとは異なり大容量メモリを搭載するCPUノードに関する設計の技術的選択肢であり、第一候補の選定期限は2027年3月末である。HPCアプリやAIフレームワークなどに関して、標準的な計算ノードよりも大容量のメモリを搭載するCPUノードを必要とするものはあるか、を問うている。すなわち、Item 1-2のメモリ容量では不十分なアプリやAIフレームワークの有無に関する問いである。また、有、の場合には、そのようなアプリの数や想定される大容量メモリの利用方法についても、調査が必要である。

実現可能性についてはさらなる技術的評価が必要であるが、現時点では、例えば以下の2通りのメモリ容量が可能なCPUノードを想定している。

- doubled memory capacity in 4x CPU sockets, or
- quadruple memory capacity in 2x CPU sockets

基本設計においては大容量メモリノードのメモリ容量例が提示されたのみであり、今後引き続きコデザインを実施する。

3.1.5 CPU および GPU の性能モデルとメモリ性能特性の調査

富岳NEXTの設計においては、CPU、GPU、およびCPUとGPUから構成される計算ノードに対してアプリケーションの性能を推定することが重要である。アプリケーションの性能推定に向けて、性能モデルの調査と検証、および、メモリなどの基本構成要素の性能特性の調査を行った。

現行CPUである「富岳」のA64FX CPU（以下、当該CPU）に対して、アプリケーションソフトウェアの計算処理性能を推定するための性能モデルを調査し、当該CPUと異なるCPUに対して理研の提供するベンチマークプログラムのいくつかを用いて、候補となるモデルの正しさを検証した。また、現行のGPU（以下、当該GPU）に対して、アプリケーションソフトウェアの計算処理性能を推定するための性能モデルを調査し、当該GPUと異なるGPUに対して理研の提供するベンチマークプログラムのいくつかを用いて、候補となるモデルの正しさを検証した。

加えて、CPUおよびGPUそれぞれのメモリ性能特性, CPU/CPU, CPU/GPU, GPU/GPU間それぞれのメモリ性能特性を調査すると共に、そのメモリ特性モデルの調査と検証を行った。

3.1.5.1 CPU の性能モデル

当該CPUを用いて得られたアプリケーションのプロファイリング情報を基に、対象とする別のCPUにおける性能を推定するモデルの構築を行った。CPUの性能モデル候補として、以下を検討した。

まず、プロファイラにより、当該CPUに対するサイクルアカウティング情報 (cycle account execution time)を取得する。次に、サイクルアカウンティング情報の各要素（命令コミット時間、L1Dキャッシュアクセス待ち時間、L2キャッシュアクセス待ち時間、メモリアクセス待ち時間、など）のそれぞれに対応する係数を掛けて、対象CPUにおけるコア単位の実行時間を推定する。次に、対象CPUでアプリケーションを実行する際のコア数を用いて、並列処理効率を考慮した実行時間を算出する。最後に、Rooflineモデルなどを考慮して、推定性能が現実的な値となるように、実効性能の上限を補正する。

コア単位の実行時間を推定するのに使用した係数として、例えば以下を考慮した。

- CPU 周波数の比率
- 浮動小数点演算パイプライン数の比率
- SIMD 長(SIMD 化率も考慮)の比率
- 浮動小数点演算レイテンシの比率
- 整数演算レイテンシの比率
- L1 キャッシュアクセスレイテンシの比率
- L2 キャッシュアクセスレイテンシの比率
- メモリアクセスレイテンシの比率

3.1.5.2 GPU の性能モデル

当該GPUを用いて得られたアプリケーションのプロファイリング情報を基に、対象とする別のGPUにおける性能を推定するモデルの構築を行った。GPUの性能モデル候補として、以下を検討した。まず、性能モデルでは、NVIDIA Nsight Computeで取得できる以下の指標を用いる。

- アクティブワープ数（smsp_warps_active.avg.per_cycle_activ）
- 整数演算命令の発行数
- LoadStore 命令の発行数

- 単精度演算系の命令の発行数
- 倍精度演算系の命令の発行数
- これらの情報から求められるそれぞれの命令の発行数の比
- メモリバンド幅/キャッシュのヒット率情報（これより、Load がキャッシュからか主記憶からかを判断する）
- SM 数
- アクティブサイクル数
- SM 内の 1 サイクルあたりの命令数

これらに基づき、命令の各種類に応じた遅延を考慮して実行サイクル数を求めるモデルを構築した。

3.1.5.3 性能モデル候補による CPU および GPU の性能推定とその検証

モデル構築や評価に用いるアプリケーションは、計算主要部と非主要部から構成される。計算主要部は、#ifdef により CPU または GPU を選択して実行する実装と、OpenACC により記述された実装のいずれか、またはそれらの併用によって構成されている。このため、計算主要部は CPU および GPU の双方で動作可能である。データ取得用の測定では、A64FX CPUを用いて計算主要部および非主要部の両方の実測データを取得する。また、検証用 GPU とは異なる A100 GPUを用いて、計算主要部の実測データを取得する。これらの測定データを用いて、CPU 性能推定モデルにより計算主要部および非主要部の性能を推定し、GPU 性能推定モデルにより計算主要部の性能を推定する。

検証用の測定では、GRACE-HOPPERのCPU 部において計算主要部および非主要部の実測データを取得し、GRACE-HOPPERのGH200 GPUにおいて計算主要部の実測データを取得する。得られた実測データと推定結果を比較することで、計算主要部および非主要部の性能推定の妥当性を検証する。また、計算主要部についてはGPU を含めた推定結果と実測データの比較により検証を行う。

CPUの性能推定と検証には、次の3つのアプリケーションを用いた。

- FrontFlow/Blue
- GENESIS
- SCALE

これらのアプリケーションに対してA64FX CPUで取得したプロファイリングデータからGRACE CPUの性能を推定し、それをGRACE CPUでの実測性能と比較し検証を行ったところ、推定実行時間が実測値より長い場合と短い場合が確認された。推定実行時間が実測値より短い遅い場合については、メモリアクセス待ちや浮動小数点演算待ちの係数を変更することで、実測値に近い値になることを確認した。今後は、キャッシュアクセス状況や、命令スケジューリング状況を把握し、係数を決定する仕組みを検討し、精度を高める必要がある。また、推定実行時間値が実測値より長い場合については、GRACE CPUに対する最適化が影響している可能性があり、コンパイラを統一することで精度を向上できる可能性が考えられる。これらの最適化やさらなる解析は、今後の課題である。

GPUの性能推定と検証には、次の3つのアプリケーションを用いた。

- FrontFlow/Blue
- GENESIS
- SCALE
- これらのアプリケーションに対して A100 GPU で取得したプロファイリングデータから GH200 GPU の性能を推定し、それを GH200 GPU での実測性能と比較し検証を行った。

3.1.5.4 メモリ性能特性

メモリ性能特性のモデル構築にあたり、以下のメモリ性能を実測した。その際、連続、ストライド、ランダムのアクセスパターンを用いた。また、実測には表 3-6に示すマシンを使用した。

- CPU-ローカルメモリ
- CPU-CPU(別サブノードメモリ)
- GPU-ローカルメモリ
- CPU-GPU
- GPU-GPU

測定データに対し、理化学研究所の提示した幾つかのモデル式を用いたフィッティングを行い、特定パラメータの抽出及びモデル式の妥当性検証を行った。その結果、それぞれのメモリ性能に対し、10%未満の目標とする精度を十分に下回る平均二乗相対誤差（MSRE）で、性能特性をモデル化することができた。

表 3-6 メモリ性能測定環境の諸元

使用環境	富岳	R-CCS		
		GH200(qc-gh200)		A100(qc-a100)
種別	A64FX	Grace	H100	A100
コア	48(12/CMG)	72	-	-
L1 サイズ	64KB/4way	128KB/コア	256KB/SM	192KB/SM
L2 サイズ	32MB/16way (8MB/CMG)	1MB/コア	50MB	40MB
L3 サイズ	-	114MB	-	-
メモリサイズ	32GB (8GB/CMG)	480GB	96GB	80GB
メモリスループット	1,024GB/s (256GB/s /1CMG)	384GB/s	4,000GB/s (NVLink-C2C: 900GB/s)	2,039GB/s (NVLink： 600GB/s)
キャッシュラインサイズ	256B	64B	128B	128B
コンパイラ	FCC-4.12.1	GCC-11.5.0	NVHPC 25.9	NVHPC 25.7

3.1.6 通信パターンの調査

Scale-outネットワークの性能推定や設計へのフィードバックを目的として、重要なベンチマークアプリケーションが要求する通信パターン情報の収集と、その解析のための環境構築を行った。スーパーコンピュータ富岳上でベンチマークアプリケーションを実行し、MPIトレースにより通信パターン情報を収集した。また、得られた通信パターン情報に基づき、各ノードの通信状況を時系列に可視化するツール、および各ノードがどのような通信をどれだけ行ったのかを時間積み上げグラフにより可視化するツールを作成した。

富岳で実行したベンチマークアプリケーションのMPIトレースは、Structural Simulation Toolkit (SST) と Score-P の2つのツールにより行った。SSTではDUMPIフォーマットによるバイナリトレースファイルが、Score-PではOTF2ファイルがMPIトレースデータとして得られる。これらのファイルから、各ノードの通信状況を読み取り、グラフ形式で可視化するツールについても整備を行った。

通信トレースを行ったベンチマークアプリケーションを表 3-7に示す。GENESISのみScore-PによりMPIトレースを行い、それ以外はSST_DUMPIによるトレースを実施した。今後、Scale-outネットワークの設計へのフィードバックのために、アプリケーションの通信パターンをベースにしたシミュレーションやモデル化による検討を実施する。

表 3-7 通信パターンをトレースしたアプリケーションとその実行サイズ

アプリケーション名	説明	ノード数	プロセス数
SALMON	光と物質の相互作用をターゲットにした第一原理計算アプリケーション	48	192
FlontFlow/blue	非圧縮流体の非定常流動を高精度に予測可能な LES に基づいた汎用流体解析コード	768	3072
GENESIS	細胞環境を想定した系の高速シミュレーションが可能な超並列分子動力学計算ソフトウェア	4	16
LQCD-DWF-HMC	DWF を用いた格子量子色力学(LQCD)のハイブリッドモンテカルロシミュレーション(HMC)	2048	8192

3.1.7 CPU 部-加速部間高速リンク接続の評価検証

CPU部-加速部間の高速リンク接続方式の基本設計を進めた結果、NVLink-C2C（NVIDIAの提案するNVLink Fusion技術）を有力候補として検討することに決定した。それに伴い、来年度実施予定の詳細設計開始に先立ち、CPU部-加速部間の高速リンク接続方式に関する検討を加速させ、当該プロジェクトの達成確度を高めることを目的として、NVLink Fusion案の評価・検証を実施した。

3.1.7.1 CPU 部-加速部間高速リンク接続の要件

以下の仕様に対し、CPU部-加速部間の高速リンク接続の評価検討を実施する。

1. CPU部と加速部間でキャッシュのコヒーレンスをとる。
2. プロトコルは、NVLink Fusionに準拠する。
3. 帯域は、NVLink-C2C に準拠する。
4. CPU部として、「富岳NEXT」基本設計で提案の、富士通製CPU「FUJITSU-MONAKA-X」を想定する。
5. 加速部として、「富岳NEXT」基本設計で提案の、NVIDIA社製GPUを想定する。

3.1.7.2 評価検証項目

以下の項目について評価検証を実施した。

まず、論理的接続に関して、前述の仕様に基づき、CPU部と加速部を接続する論理的な接続方式について以下の検討を行い、アーキテクチャを確定した。

- ノード当たりの GPU ソケット数
- CPU-GPU 接続方式(プロトコル,データリンク,PHY,レーン数,周波数,キャッシュコヒーレンス制御等)
- NVLink Fusion CPU 側論理仕様
- FUJITSU-MONAKA-X ARM CHI 論理仕様との整合性
- NVLink Fusion に付随する GPU とのインターフェース仕様

また、論理的な接続方式を評価、検証する方法として以下の検討を行った。

- VIP によるソフトシミュレーション方式
- エミュレータによる高速シミュレーション方式

次に、物理的設計に関して、前述のCPU部-加速部間の高速リンク接続仕様に基づき、CPU部と加速部の接続におけるLSI、パッケージ、マザーボード、周辺回路部品の物理的な接続方式について以下の検討を行った。

- LSI のフロアプラン
- LSI 及びパッケージのピンアウト
- パッケージ及びマザーボードのクロスセクション

また、物理的な接続方式を評価、検証する方法として以下の検討を行った。

- シグナルインテグリティの検証方式
- パワーインテグリティの検証方式

最後に、半導体テクノロジに関して、検討した論理的接続および物理設計を実現するために必要となる半導体

とIO回路について以下の調査、検討を行い、採用する半導体テクノロジノードとIO回路のIPを確定した。

- NVLink Fusion に最適な半導体テクノロジの Spec
- NVLink Fusion 物理 PHY の GRS（Ground-Referenced Signaling）
- NVLink Fusion に付随する GPU 接続の物理 PHY

また、これらに関して、回路の物量、面積、電力、性能などの見積もりを行った。

3.2アプリケーション開発

3.2.1 開発体制と計画

3.2.1.1 概要、方針

システム基本設計において、富岳NEXTを活用し成果を創出しうるアプリケーションの選択、開発、チューニングを行う。また、その過程でシステム設計パラメータ決定へのフィードバックを行う。これら一連の業務を設計開発プロジェクトにおけるシステムコデザインの観点で実施していく。

少数の限られたアプリケーションによるコデザインに陥ること無く、幅広く多くのアプリケーションの開発を促し、必要に応じてそれらをコデザインに活用するためのフレームワーク開発を並行して進める。これらフレームワークを順次コデザインに活用しフレームワークの高度化も同時に進めていく。

このような開発を進めていくための組織として理化学研究所 次世代計算基盤開発部門次世代計算基盤アプリケーション開発ユニットのもとにアプリケーションエリアを設置し、アプリケーションコミュニティ、ベンダとの連携、アプリケーションの共同開発、ベンチマーク、富岳NEXTを想定した性能推定を行う。

アプリケーションエリアの構成として、アプリケーション分野ごとにサブワーキンググループを設け、それらを統括する組織としてアプリケーションワーキンググループを設置する。アプリケーションの評価、ベンチマーク、それらに用いる継続開発環境の開発を行うベンチマークワーキンググループを設置し、エリアとしては二つのワーキンググループ体制で開発を行う。

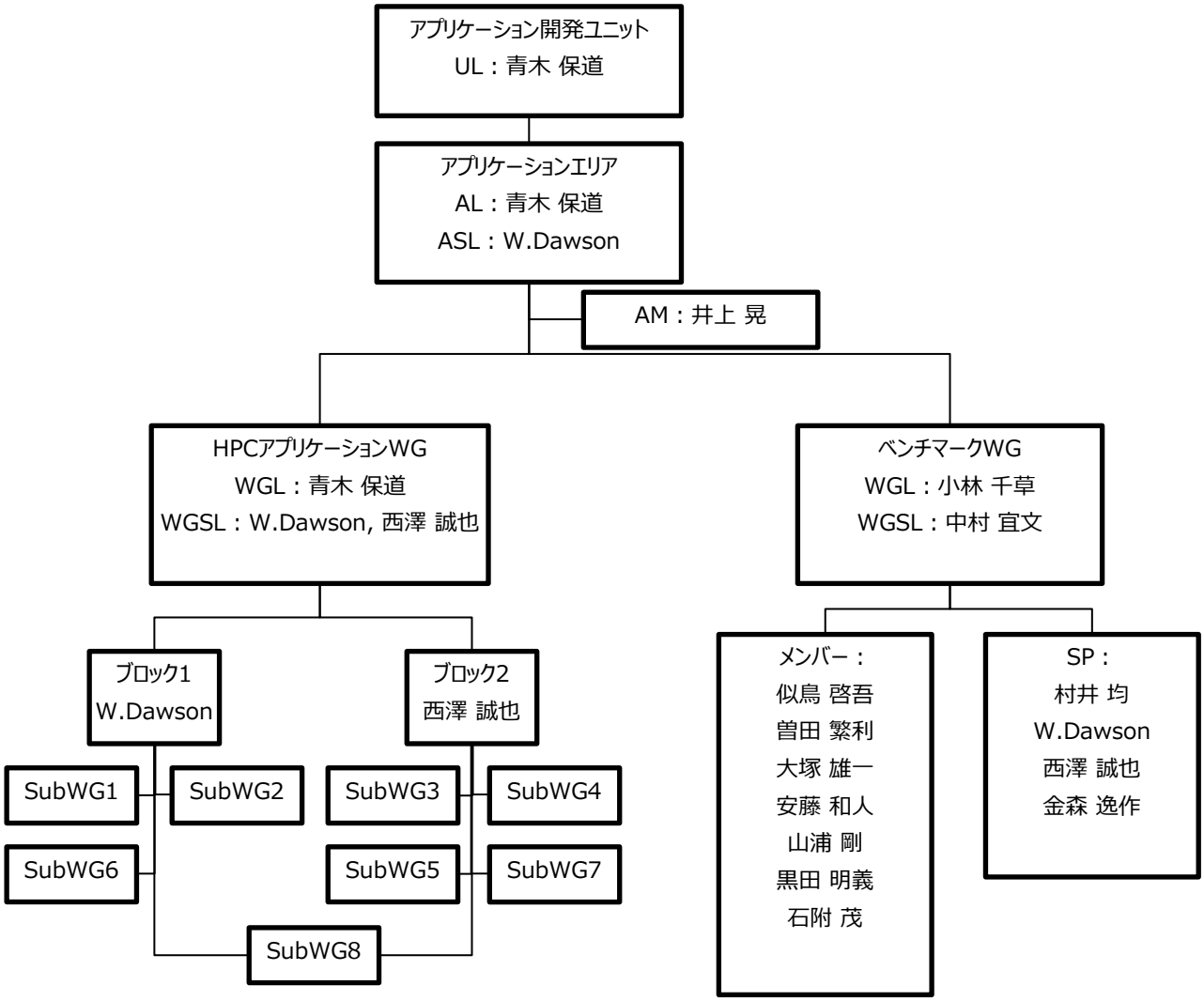
これらの体制構築において2024年度まで理化学研究所を中心に行われてきた - ポスト「富岳」FS（フィージビリティスタディ）のアプリケーション調査グループの組織構成を引き継ぎつつ、一部活動実績に応じて修正し、アプリケーションサブワーキンググループ構成を行った。FSベンチマークグループは、その活動重要性と負荷の度合いを鑑み、ワーキンググループに昇格する事とした。

FSにおいて評価対象となったアプリケーション分野、および分野代表の選出は、HPCIコンソーシアム計算科学フォーラムがまとめた「計算科学ロードマップ2023」(https://cs-forum.github.io/roadmap-2023/)に基づきアプリケーション分野に問いかけて行われた。当基本設計においては、各サブワーキンググループの統括を、少数の例外を除いて、理研R-CCSメンバーを新たに充て、アプリケーションコミュニティとの連携はFSの分野代表の方々に引き続き努めて頂く方針をとる。また、分野ごとにR-CCS内から必要に応じて参加メンバーを追加する。これにより、コミュニティの連携を図りつつ、他エリアとの部門内での密な連携が可能となり、結果として効率的な業務遂行が可能となる。

3.2.1.2 体制

- アプリケーションエリア – ワーキンググループ(WG)、サブワーキンググループ(SubWG)

アプリケーション開発ユニットの実施体制を示す。※敬称略



HPCアプリケーションWGは下記のブロックとサブWGで構成される。

ブロック	体制
ブロック１	BOG：William Dawson AD：深沢 圭一郎、中島 研吾
ブロック２	BOG：西澤 誠也 AD：岩下 武史、富田 浩文

SubWG	体制
①生命科学 【杉田チーム、TAMA チーム】	OG：伊東 真吾 LS：寺山 慧 （Wenyang Zhao）

②新物質・エネルギー 【中嶋チーム、柚木チーム】	OG：William Dawson LS：山地 洋平 幸城 秀彦、（Enhua Xu）
③気象・気候 【富田チーム、三好チーム】	OG：大塚 成徳 LS：小玉 知央 足立 幸穂、河合 佑太、雨宮 新、James Taylor、垂水 勇太、（Unashish Mondal）、西澤 誠也
④地震・津波防災 【富田チーム】	LS/OG：藤田 航平、(西澤 誠也)
⑤ものづくり 【坪倉チーム】	OG：大西 順也 LS：高木 亮治 AD：加藤千幸
⑥基礎科学 【青木チーム】	OG：金森 逸作 LS：滝脇 知也、金森 逸作
⑦スマートシティ、デジタルツイン・Society5.0 【山口チーム】	LS/OG：山口 弘純 AD：下川辺 隆史 田中 福治
⑧科学技術計算・機械学習アルゴリズム 【今村チーム】	OG：鈴木 厚 LS：高橋 大介 横田 理央

なお、本ユニットの役割として下記を設定する

UL	ユニットリーダ	AL	エリアリーダ
ASL	エリアサブリーダ		
WGL	WG リーダ	WGSL	WG サブリーダ
AD	アドバイザ	BOG	ブロックオーガナイザ
LS	リエゾン	OG	オーガナイザ
SP	協力者	AM	エリアマネージャ

役割分単

役割	担務
UL	本ユニットの責任者
AL	エリアの責任者
AM	本エリア全体のプロジェクト管理 本エリア外部との連携調整 全体会議の議事録作成
WGL	WG の責任者 WG 体制の管理 開発検討会議の運営

WGSL	WG のプロジェクト管理 開発検討会議の議事録作成
BOG	ブロック会議の運営
AD	アプリケーション開発のアドバイザ
OG	SubWG の運営 SubWG 体制の管理 検討会議の議事録作成
LS	開発コミュニティとの連携

- アプリケーションワーキンググループ

アプリケーションコミュニティと連携したアプリケーションの選択に始まる開発はこの組織(理研プロジェクトメンバー)を中核、もしくは窓口として、アプリケーション開発者と協力体制の元行われる。開発業務はサブワーキンググループ(SubWG)の単位で行われるが、ここでの情報伝達、情報の共有の単位として4つのSubWGをまとめたブロックを構成し、ブロックリーダーによる開発業務管理が行われる。ブロック1は SubWG 1, 2, 6, 8 からなり、ブロック2は SubWG 3, 4, 5, 7 (,8) で構成される。SubWG 1～7はHPCアプリケーション分野であり、SubWG 8 は HPCで用いられる数値アリゴリズム(機械学習を含む)を担当する。各SubWGには統括を行うオーガナイザ(OG), 開発コミュニティとの連携を行うリエゾン(LS)を置く。OGは原則R-CCS計算科学チームの常勤メンバーから専任し、LSは原則としてフィージビリティスタディ(FS)で分野代表を担っていた方を選任する。

後述する、アプリケーションコミュニティへのアンケート調査は各ブロック→SubWG→コミュニティの開発者のラインで展開され行われる。

今年度初期評価対象アプリケーション(EEA, Early Evaluation Application)を選出し、その中から6つについてはベンダとの共同作業を開始した。これら6つのアプリケーショングループをEEA1と表記する。

- ベンチマークワーキンググループ / Benchmark WG

当WGはアプリケーションソフトウェアの開発、ベンチマーク、性能推定プロセスを統一して扱うフレームワークの開発を行い、プロジェクト内外に公開する業務を担う。アプリケーションワーキンググループと密に連携を図り、評価対象アプリケーションのフレームワークへの取り込み、ベンチマークの実施にも関わる。今年度各EEAにベンチマークWG担当者を割り当て開発サポートを実施した。また、フレームワークとして用いる Bench Kit の開発、Benchpark 活用方法の検討及びハッカソンの実施をコデザインWG、Lawrence Livermore National Laboratory (LLNL) の Benchpark 開発者と連携して行った。

- ベンダとの連携

今年度基本設計において、アプリケーション開発におけるベンダとの接点はコデザイン要素技術検討会に集約される。当要素技術検討会は、理研、富士通、NVIDIAの3者からなり、月1回のペースで計8回開催された。それぞれの検討会の議題の頭出しや深掘りを行うためのSync-Upミーティングを各要素技術検討会前に理研-富士通、理研-NVIDIAの2者会としてそれぞれ開催した。これらの会議では、アーキテクチャからのコデザイン要素及び選択肢の提示と検討、アプリケーション開発の進捗、アプリケーションエリアにおける調査報告など、コデザインに資する業務の確認及び決定、また両ベンダと理研が共有すべき情報の交換が行われた。理研からは、主に アプリケーションエリアとアーキテクチャエリアが参加し、アプリケーションエリアリーダが議長を務めた。

- 他エリアとの連携

ベンダとの連携で述べたように、アプリケーションソフトウェア、アプリケーション開発者からのフィードバックに基づくコデザインは、アプリケーションエリアとアーキテクチャエリアとの連携を主として行われた。アーキテクチャからのコデザイン要素及び選択肢の提示に基づき、アプケーションがからのフィードバックを取りまとめ共有した。また、アプリケーションが用いるプログラミング環境、AIを活用したコーディングに関する情報共有を、当該環境の基本設計を行うシステムソフトウェアWGに対して行った。次年度以降、特にプロジェクト内外のアプリケーション開発者への開発環境提供において運用技術エリアとの連携が重要になる。図 3-11は今年度のエリア間連携を模式的に表したものである。

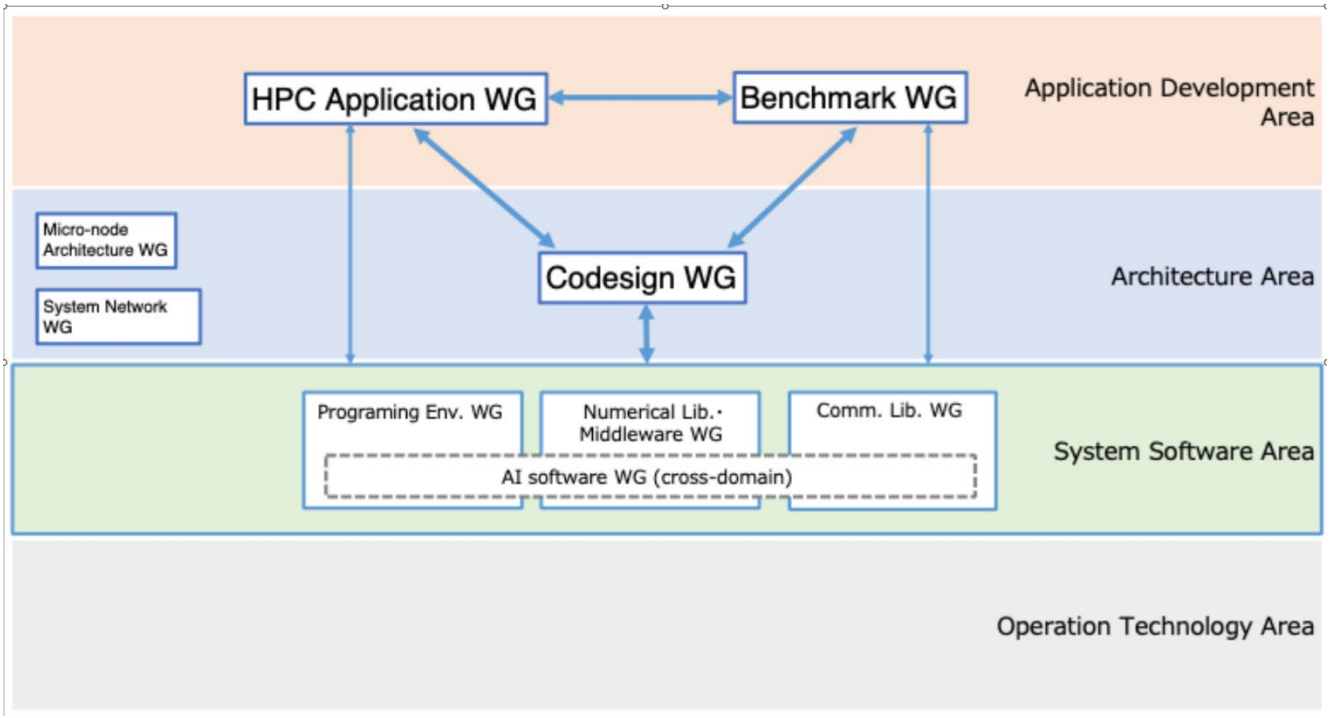


図 3-11 今年度のエリア間連携

3.2.1.3 開発計画

- 2025 年度

富岳NEXTアプリケーション開発における2025年度の開発項目は以下のとおりである。項目と関連するセクションも併記した

A: 開発環境構築とアプリケーション登録 (3.2.2)

(1) 継続ベンチマーク開発環境の構築

(2) アプリケーションの環境登録

B: アプリケーション開発とコデザイン (3.2.3, 3.2.5)

(3) 初期評価対象アプリケーションの選択

(4) 評価、コデザイン方針の決定

(5) 今年度評価対象アプリケーションの絞り込み

(6) アプリケーション初期性能評価

(7) アプリケーションGPUチューニング

(8) アプリケーションの現行機による評価

(9) 富岳NEXTをターゲットとした性能推定

C: アプリケーションベンチマークに寄らないコデザイン (3.2.4)

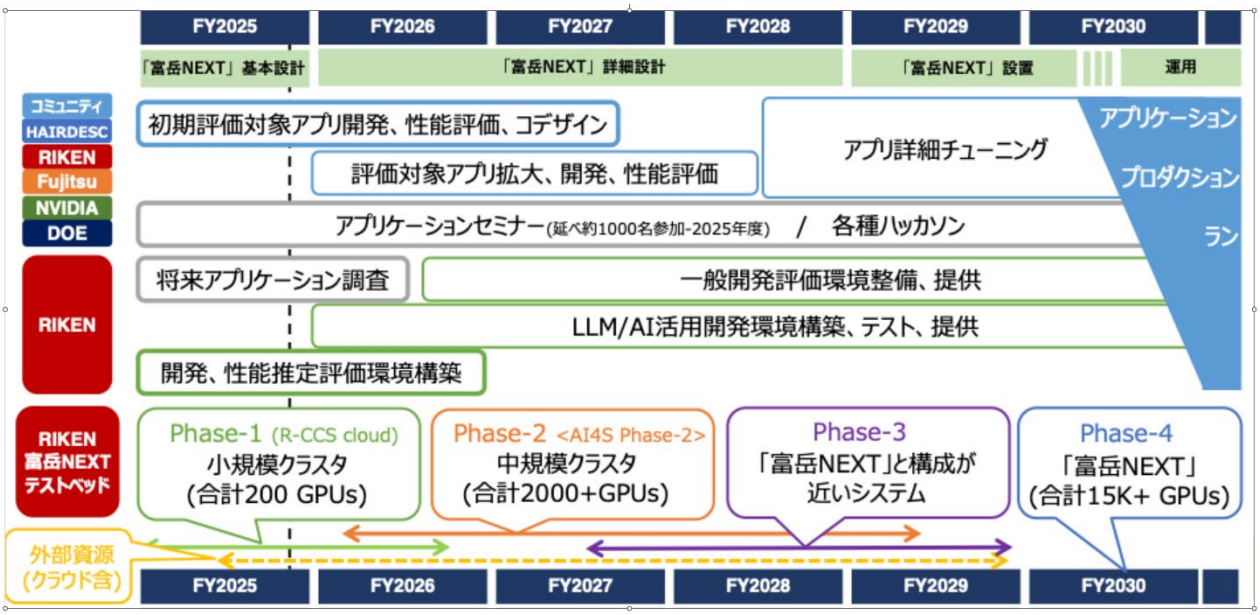
(10) アプリケーションコミュニティへの調査に基づくシステム設計項目へフィードバック

(11) 富岳運用統計情報解析からのシステム設計項目へフィードバック

D: 富岳NEXT時代にむけた 新しいアプリケーションの開発 (3.2.6)

- 富岳 NEXT 運用開始までの計画

富岳NEXT運用開始までのアプリケーションエリアの開発計画、及び、関連するテストベッドの導入計画を以下に示す



3.2.2 アプリケーション開発環境

3.2.2.1 開発計画

この章では、アプリケーション開発・実行・性能評価を一体として捉え、アプリケーション開発環境の観点から検討および整備した内容について述べる。

基本設計、詳細設計を通した協調設計のためには、各種ハードウェア要件の検討に資する定量的な情報を提供することが求められる。さらに次世代機の性能を最大限に引き出すべく最適化されたアプリケーションコード（アルゴリズム）の開発を通して、実行性能の観点から評価を行う。このようなハードウェア要件の検討・評価のためには、アプリケーションの計算特性やボトルネックを把握すべく詳細なアプリケーション性能評価が不可欠となる。その際、評価結果の比較及び再現性の確保のためにアプリケーションだけでなく、ライブラリ・コンパイラのバージョン、ビルド条件を明確に管理した状態で性能評価を行う枠組みが必要となる。性能評価を体系的に実施するためのベンチマークフレームワークの構築を計画し、本プロジェクトの用途に適したフレームワークの設計方針を策定した。また、特定のベンダに依存しない継続的な環境を構築することも重要であることから、国際的に利用されているオープンソースソフトウェア(OSS)を積極的に活用した開発・実行環境の構築を基本方針とした。ただし、アプリケーション開発者の参入障壁やアプリケーションの整備状況を鑑みて、この基本設計段階では一つのフレームワークに集約するのではなく、運用特性の異なるフレームワークを組み合わせることで、より広い評価アプリケーションの参加を促した。フレームワークを通すことで開発によるコードの更新と性能値が１対１の関係で得られることとなる。そのため、これらのデータは、GPUコードへの変換などのAIを通した開発のための学習データになり得る。詳細設計期間では、これらのコードの更新と性能値を学習データとして使えないかも検討をする必要がある。

このフレームワークは、後述する早期評価用アプリケーション(EEA)だけでなく、システムソフトウェアで検討されているアプリケーションやAIアプリケーションに対しても性能評価に利用される予定である。また、性能評価方法に関しても、既存の手法だけでなく、システムソフトウェアで開発された評価ツールも組み入れて利用されることを想定している。

現在は、14のEEAを中心に開発・性能評価を行っているが、詳細設計に向けてはより幅広いプログラミングモデル（GPU-resident modelなど）、並列モデルを持つソフトを加えることが必要となる。

3.2.2.2 Benchpark

Benchpark は、ローレンス・リバモア国立研究所（LLNL）を中心に開発が進められている HPC 環境向けアプリケーションベンチマークの自動化を目的とした Python ベースのオープンソースフレームワークである。ソフトウェアのビルドには Spack を、実行ワークフローの管理および結果収集には Ramble を用いる構成を採用しており、コンパイラや MPI を含むソフトウェアスタックを明示的に定義し、再現可能な形でベンチマークを実行できる点に特徴がある。この構成により、依存関係を含めた厳密な再現試験や長期的なリグレッション検証を実施することが可能であり、ストロングスケーリングおよびウィークスケーリング試験を体系的に行う基盤として有効である。また、CI 機構と連携することで、ビルドおよび基本的な実行検証を継続的に実施することができる。 一方で、Spack を前提とした環境構築は依存解決や多数のファイル生成を伴うため、構成規模や利用環境によっては環境構築に時間を要する場合がある。特に Lustre 上ではメタデータ操作の集中により構築時間が増大する傾向がある。ただし、Spack 自体の性能改善は継続的に進められている。 本基盤においては、詳細な依存関係管理や高い再現性が求められる評価に対して Benchpark を活用することを想定する。

3.2.2.3 Benchkit

Benchkit は、Bash を基盤とする軽量なベンチマークフレームワークであり、ベンチマーク実行、CI/CD 制御、ならびに複数拠点で実行された結果の集約を可能とする。本基盤では、継続的評価の入口として Benchkit を位置付ける。Benchkit では、ベンチマーク実行後に JSON 形式の結果ファイルや詳細な Performance Analysis（PA）情報をアップロードすることができ、集約ページにおいて必要な値を抽出・整理・可視化する仕組みを備える。また、性能推定モデルを差し替える構成により、例えば 富岳 を基準とした性能倍率算出などの比較評価を実施できる。 これらの処理は CI/CD の仕組みによりパイプライン化されており、実行から結果集約、性能推定、公開までを継続的に実施できる構成となっている。Bash ベースであることから、既存のビルドスクリプトや実行スクリプトを大きく変更することなく利用でき、外部アプリケーションや多拠点参加者にとって参入障壁の低い構成となっている。

本基盤では、軽量な継続的評価および性能集約を担う層として Benchkit を活用し、依存関係を含めた厳密な再現性が求められる評価については Benchpark を併用する構成を想定する。Benchkit から Benchpark を呼び出して実行することも可能であり、両者は補完的な関係にある。今後の検討および実装の進展に応じて、性能集約機構や推定機構などの有用な要素については、Benchpark 側への統合や相互連携の高度化を段階的に検討する。これにより、運用効率と再現性の双方を両立する拡張可能な開発基盤の構築を目指す。

3.2.2.4 性能評価方法

性能評価については、継続的ベンチマーク実行と性能推定・解析を疎結合に接続する拡張可能な構造を基本方針として検討している。現時点では、その実装例として Benchkit を活用しているが、本設計は特定の実装に固定されるものではなく、今後の検討および運用を通じて適切な構成を具体化していくことを想定している。 ベンチマーク実行により生成される結果データについては、軽量なメタデータを保持する JSON 形式のファイルと、詳細な性能解析情報を含む TGZ 形式のアーカイブファイルとを分離して管理する方式を基本案としている。両者を同一の UUID により関連付けることで、最小限のメタデータ管理と解析用大容量データの柔軟な保存を両立できる可能性がある。この UUID を基点として、性能推定や追加解析のための各種パイプラインへ接続する構成を検討している。 性能推定機構については、ベンチマーク実行基盤本体とは分離した構成とすることを想定している。特定の性能モデルや推定手法を基盤内部に固定的に組み込むのではなく、外部ツールとして実装された推定モジュールを CI/CD パイプラインの一部として呼び出す方式が有力な選択肢である。このような分離設計を採用することで、アプリケーションごとに異なる推定手法の適用や、性能モデルの更新・差し替えを柔軟に行えると考えられる。また、性能推定モデルが機密情報を含む場合を想定し、推定ロジックや内部パラメータを外部に公開しない運用形態も考慮する。 さらに、ベンチマーク結果および性能推定結果の公開範囲についても、アプリケーション単位で柔軟に制御可能とする方向で検討している。生データの非公開設定や認証付き公開など、情報公開の制約に配慮した運用形態に対応できる構成を目指す。 本機構はあくまで基本設計段階にあり、今後の実装および運用状況を踏まえて段階的に詳細化していく予定である。

3.2.2.5 開発環境利用の実際

本節では、整備したアプリケーション開発環境および性能評価フレームワークを用いて、実際にアプリケーションがどのように導入・構築・評価されるかについて、その一連の流れを図 3-12に示す。

まず、図の左上に示すように、各アプリケーションのソースコードおよび入力データは、インターネット上のリポジトリ、

もしくは内部サーバ（非公開アプリケーションの場合）において、バージョン管理システムを用いて管理される。

ベンチマークフレームワークは、必要に応じてこれらの情報を取得し、各計算機システム上でアプリケーションのビルド、実行、および性能評価を行う。実行および性能評価の結果は、性能評価用のデータベースサーバへ自動的に送信され、プロジェクト参加者はこれらの評価結果を共有・確認することができる。この仕組みにより、評価対象となるソフトウェア構成を明確に管理した状態で、継続的かつ再現性のある性能評価を実施することが可能となる。

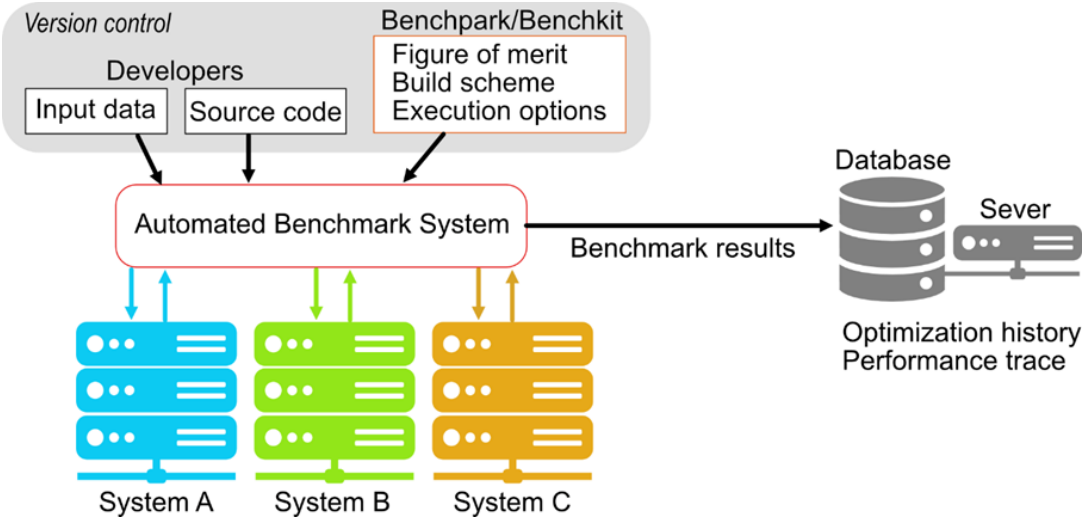


図 3-12 ベンチマークフレームワーク概念図

3.2.2.6 開発環境公開にむけて

本プロジェクトで構築した性能評価フレームワークについては、すでに外部公開を行っており、アプリケーション性能評価を実施するための基盤として利用可能な状態にある。

- Benchpark https://github.com/RIKEN-RCCS/benchpark,
- Benchkit https://github.com/RIKEN-RCCS/benchkit

一方、性能評価結果として蓄積されるデータの公開方法については、利用形態や公開範囲を考慮する必要があることから、詳細設計において検討が必要である。

また、評価対象となるアプリケーションのソースコードについては、多くのソフトウェアがオープンソースソフトウェア（OSS）であることから、すでに公開されているものが多い。一方、入力データについては、今後の研究内容や計画に関わる機微な情報を含む場合があるため、ソースコードとは個別に取り扱う必要があるとの意見が利用者から寄せられている。このため、入力データの公開方法については、今後の詳細設計において検討を行う必要がある。

さらに、非公開アプリケーションについては、著作権者との契約条件や安全保障貿易管理等の関連法令を考慮する必要があることから、現在、各開発者と協議を行っている段階である。

3.2.3 アプリケーション開発

3.2.3.1 開発対象アプリケーションの選定方針

本プロジェクトにおけるアプリケーション開発は、富岳NEXT のアーキテクチャ設計に対して実効的なフィードバックを与えることを主目的とする。そのため、単なる移植性や既存性能の延長ではなく、将来の大規模科学計算を牽引する代表的アプリケーション群を選定対象とした。選定の基本方針は以下の通りである。

- 国内研究コミュニティにおいて重要性が高いこと
- FugakuNEXT において科学的ブレイクスルーを目指す規模・複雑性を持つこと
- 計算特性が多様であり、設計上のトレードオフを顕在化させること
- 将来的な上流マージを見据えた開発体制が構築可能であること

3.2.3.2 早期評価用アプリケーション（Early Evaluation Applications, EEA）

(1) 概要

早期評価用アプリケーション（Early Evaluation Applications, EEA）は、アーキテクチャ設計初期段階において性能特性・メモリ要求・通信特性を明確化するための中核アプリケーション群である。

選定されたアプリケーションは以下の通りである。

名前	SubWG	種類	計算特性分類
Genesis	SubWG1	分子動力学シミュレーション	3
UT-Heart	SubWG1	心臓電気生理・力学連成シミュレーション	5 / 6
SALMON	SubWG2	第一原理電子状態計算（TDDFT）	5
mVMC	SubWG2	量子モンテカルロ法による強相関電子系シミュレーション	1
SCALE-LETKF	SubWG3	気象シミュレーションおよびデータ同化（局所アンサンブルカルマンフィルタ）	1, 4, 5
E-Wave	SubWG4	地震波伝播シミュレーション	6
FrontFlow/blue	SubWG5	非構造格子有限体積法による流体解析（CFD）	6
FFVHC-ACE	SubWG5	非構造格子有限体積法による高忠実度流体解析	5
LQCD-DWF-HMC	SubWG6	格子QCDシミュレーション（Domain-Wall Fermion, HMC）	5
MHDTurbulence	SubWG6	磁気流体乱流シミュレーション	5
Sionna + SUMO	SubWG7	デジタルツイン・大規模社会シミュレーション	5, 6
PETSc	SubWG8	大規模並列線形ソルバ基盤	
PyTorch/Megatron	SubWG8	大規模言語モデル学習（分散深層学習基盤）	

(2) GPU対応状況と計算特性分類

各EEAについて、GPU対応状況、使用プログラミングモデル、計算特性（B/F比、メモリアクセス形態、演算密

度）を整理した。

計算特性は以下の観点で分類した。

1. 行列積型（高演算密度）
2. 低B/F・単純ループ型
3. キャッシュブロッキング可能型
4. 複雑ループ構造を持つ低B/F型
5. 高B/F・連続メモリアクセス型
6. 高B/F・間接メモリアクセス型

その結果、多くのアプリケーションはメモリ帯域律速型または通信律速型であり、純粋な計算律速型は限定的であることが明らかとなった。詳細設計においては、より広範な計算特性をもつアプリケーションセットの構築が課題である。

3.2.3.3 安全保障輸出管理

オープンソースソフトウェアではないアプリケーションを海外ベンダに開示する場合には、大量破壊兵器の拡散防止、通常兵器の過剰蓄積防止、技術情報流出の防止、国際社会の信頼維持のために適切に安全保障輸出管理を行う必要である。理化学研究所では研究インテグリティ・経済安全保障部が中心になり審査を実施するため、審査を受ける部門は適切な書類を用意し依頼する。審査の対象となるアプリケーションはFFB、FFVHC-ACE、E-Waveである。いずれのアプリケーションも開発主体は理化学研究所外であるが、本プロジェクトにおいて海外ベンダへアプリケーションを開示するのは理化学研究所であるため、安全保障輸出管理の審査も理化学研究所が実施する必要がある。2025年9月時点では下記の書類を用意し、担当部門へ依頼を実施した。

1. 該非判定書（技術）
   技術の名称、技術の内容、該非判定結果、判定に関する説明、判定の根拠資料に必要事項を記入する。なお、判定の根拠資料は「貨物と技術のマトリクス表」とした。
2. 審査票（技術の提供・共同研究）
   技術の名称、技術の概要、開示先の情報、最終用途、提供機関を記入する。

3.2.3.4 共同研究契約の検討

オープンソフトウェアではないプロプライエタリソフトウェアをベンダと共同で開発するには、秘密保持契約で定める秘密情報の取扱い以外に、開発により生じる知的財産権の帰属を明確にする必要がある。本プロジェクトにおいては、アプリケーション開発主体、NVIDIA、富士通、理化学研究所がそれぞれの強みを持ち寄り、これまでにないアプリケーションの開発が期待されており、これら関係者間の共同研究契約を締結し、強固な開発体制を組織することが求められている。

契約の基本骨子を下記のように定めベンダに提供し、共同研究契約締結の可否、期間、その他懸念事項などのフィードバックを得て、最終的な契約締結を目指す。対象とするアプリケーションは東京大学地震研究所が開発するE-Waveとするが、今後同様の枠組みで対象アプリケーションを増やす計画である。

1. 契約形態
   - ４者共同研究契約（東京大学地震研究所・富士通・NVIDIA・理研）
   - 各機関が強みを持ち寄り、実装面およびアルゴリズム開発を含めた研究開発を実施
2. ソースコードの開示範囲

- プロジェクト関係者（理研・富士通・NVIDIA の開発担当者）に限定して開示
- プロジェクト目的に沿った利用に限定

3. ベンチマーク性能データの取り扱い
   - プロジェクトで取得された性能データはオープンに公開
   - 学術発表、技術資料等での活用を想定
4. 知的財産権の取り扱い
   - バックグラウンド知財
     - ✧ 各機関が持ち込んだ既存の知的財産権は、それぞれが保有
     - ✧ 本共同研究期間中は、参画機関間で合意のうえ、プロジェクト内容に沿う形で使用可能
   - フォアグラウンド知財
     - ✧ 共同研究参画機関間では自由な利活用を認める（改変権利、商業実施権を含む）
     - ✧ 参画機関以外への権利許諾については、参画機関間で協議のうえ決定

3.2.3.5 テストベッド利用規約の検討

アプリケーション開発者が自由に利用可能な環境としてテストベッドと呼ばれる環境を提供予定である。この環境は理化学研究所が用意する最新のGPUや将来的にはMONAKA試作機などを装備し、富岳NEXTへのシームレスな移行を目指して準備が進められている。

テストベッドは広く利用者に公開予定であるが、ここで開発されるアプリケーションのソースコード情報をアプリケーションの高度化支援機能の学習データとして利用する。そのため、テストベッド利用者は下記の条項を含む利用規約に同意する必要がある。もちろん、ユーザは高度化支援機能を利用することが可能である。

- ソースコードを高度化支援機能の強化学習に原則利用することができる
- 高度化支援機能の高度化のために学習した AI モデルの知財は理研単独に帰属する

3.2.3.6 EEA1 初期評価用アプリケーション(1)

EEA1 (初期評価用アプリケーション第1グループ)は、EEA の中から特に早期にベンダと共同評価を実施したアプリケーション群である。選定基準は以下である。

- ライセンスや準備状況から、ベンダへ迅速に提供可能であったこと
- アプリケーション開発者とベンダとの密な技術的対話が可能であること
- GPU 化や性能解析において具体的課題が明確であったこと

すなわち、アーキテクチャ設計への即時フィードバックが期待できるものを優先した。選定されたアプリケーションは以下の6つである。

- Genesis
- SALMON
- SCALE-LETKF
- E-Wave
- FrontFlow/blue
- LQCD-DWF-HMC

NVIDIA社にこれら6本のアプリケーション、富士通社にこのうちの2本 (SCALE-LETKF と LQCD-DWF-HMC) を提供し、最適化や評価が行われた。

3.2.3.7 各 SubWG 活動概要

本節では、各SubWGから提出された個別報告書の概要を整理する。

各分野における活動内容、早期評価用アプリケーションの選定理由、計算特性、設計上の課題および今後の重要項目については、付属資料「アプリケーションWG SubWG報告」に詳細を記載している。本節では、それらの報告内容のうち、基本設計フェーズにおけるアーキテクチャ設計検討に直接関係する要点を抽出し、横断的観点から整理する。

なお、各SubWG報告書は分野ごとの科学的背景やコミュニティ活動状況も含んでいるが、本節では主として以下の観点に基づき要約を行う。

- 早期評価用アプリケーションの計算特性
- GPU 対応状況および移植課題
- メモリ容量・帯域・通信性能への要求
- 精度要求および混合精度活用可能性
- CPU-GPU 協調利用の可能性

詳細な個別報告書については付録Aを参照されたい。

- SubWG1 生命科学分野

生命科学分野では、分子動力学シミュレーションコード GENESIS および心臓マルチスケール連成解析コード UT-Heart を早期評価用アプリケーションとして選定した。両者はいずれも「京」および「富岳」での最適化実績を有し、MPI と OpenMP によるハイブリッド並列化を基本とする。

GENESIS は全原子モデル、粗視化モデル、QM/MM を含むマルチスケール MD を対象とし、一部 CUDA 実装が存在するものの、依然として計算の大部分は CPU 上で実行される構成である。粒子間相互作用計算やメモリアクセスが主要な計算コストを占め、メモリ帯域への依存が強い。今後 GPU 常駐化を進めることで、GPU メモリ容量と帯域が性能を規定する要因となる。

UT-Heart は電気生理・力学連成を含む大規模疎行列計算を特徴とし、時間発展型反復計算および大規模 I/O を伴う。

本分野からは、CPU 側性能の継続的重要性、GPU メモリ容量の十分な確保、ならびにマルチスケール対応に伴う大容量メモリ需要が設計上の重要論点として示された。

- SubWG2 新物質・エネルギー分野

本分野では、時間依存密度汎関数理論コード SALMON と、多変数変分モンテカルロコード mVMC を選定した。

SALMON は電子動力学を対象とするステンシル型計算を主体とし、メモリ帯域およびレイテンシに対する要求が高い。従来の k 点並列に基づくベンチマークではなく、スーパーセル拡大型問題設定を新たに構築し、将来の科学課題に即した計算パターンを評価対象とした。

一方 mVMC は稠密線形代数演算を多用する計算律速型アプリケーションであり、倍精度浮動小数点性能を評価する上で重要な位置付けを持つ。ただし GPL ライセンス制約および GPU 版未整備が課題である。

材料分野では、従来型第一原理計算に加え、機械学習原子間ポテンシャルなど AI 統合型計算の重要性が増している。多くの手法で倍精度が必要である一方、混合精度適用可能な領域も存在する。

設計上は、FP64 性能、メモリ帯域、ならびに混合精度演算支援が重要である。

- SubWG3 気象・気候分野

  気象・気候分野では、数値予報モデル SCALE と局所アンサンブル変換カルマンフィルタ LETKF を組み合わせた SCALE-LETKF を選定した。本アプリケーションは大規模三次元格子計算、行列演算を含むデータ同化、MPI 通信、大規模 I/O を含む複合型構造を持つ。

  SCALE 部分はステンシル型計算が支配的であり、メモリ帯域律速傾向が強い。LETKF 部分は行列演算を多用するが、GPU 移植は未完了である。

  本分野からは、メモリ帯域重視設計、行列演算支援機能の有効性検討、低精度演算適用可能性の評価が設計課題として提示された。

- SubWG4 地震・津波防災分野

  本分野では、非構造有限要素法による地震波動伝播コード E-Wave を選定した。疎行列ベクトル積および共役勾配法が主要計算であり、MPI による隣接通信と Allreduce が高頻度で発生する典型的なメモリ・通信律速型アプリケーションである。

  さらに、本分野では CPU-GPU 協調利用の実証研究が報告され、CPU メモリを活用した初期値推定により GPU 上の反復回数を削減することで、実行時間および消費エネルギーを同時に削減できる可能性が示された。

  設計上は、高帯域メモリ、低遅延通信、CPU-GPU 高速接続の重要性が示唆される。

- SubWG5 ものづくり分野

  ものづくり分野では、非構造格子 LES コード FrontFlow/blue と、構造格子 LES コード FFVHC-ACE を選定した。前者は間接メモリアクセス型、後者はステンシル型であり、異なるメモリアクセス特性を持つ。

  両者とも大規模乱流解析を対象とし、メモリ容量、帯域、ノード間通信性能が性能を規定する。GPU 実行は可能であるが、最適化余地が残されている。

  本分野からは、メモリ容量と帯域の両立、通信性能確保、GPU 前提最適化推進が設計上の重要課題として示された。

- SubWG6 基礎科学分野

  基礎科学分野では、格子 QCD コード LQCD-DWF-HMC と磁気流体乱流コード MHDTurbulence を選定した。

  LQCD はステンシル計算と反復ソルバを主体とし、global reduction が高頻度で発生する。混合精度アルゴリズムが導入されており、通信性能とメモリ帯域が支配的である。ポッドサイズが性能に影響する可能性も示唆されている。

  MHDTurbulence は三次元一様格子上の時間発展型有限差分計算であり、メモリ帯域依存が強い。Kokkos 導入による移植性向上も検討されている。

  本分野からは、通信性能、混合精度支援、GPU 間直接接続性能が重要であるとの示唆が得られた。

- SubWG7 スマートシティ分野

  本分野では、無線電場シミュレーション Sionna と Agentic AI を組み込んだ SUMO を選定した。レイトレーシングによる高分岐・不規則メモリアクセス計算と、ニューラルネットワーク推論を含む高演算密度計算が混在する構造を持つ。

  現在は要素単体での評価段階であり、MPI 並列化および統合シミュレータ構築は今後の課題である。

設計上は、分岐処理性能、GPU 演算性能、ならびに大規模並列通信性能が重要である。

- SubWG8 数値計算・機械学習アルゴリズム分野

  本分野では、疎行列ソルバライブラリ PETSc と、LLM 学習基盤として PyTorch/Megatron を対象とした。

  PETSc は SpMV を中心とするメモリ律速型ライブラリであり、GPU 混在 MPI 環境での分散計算を前提とする。前処理によって演算密度を高める可能性がある。

  一方 PyTorch/Megatron は高演算密度・混合精度前提の深層学習ワークロードであり、GPU 間高速通信が性能を左右する。

  本分野からは、メモリ帯域と演算性能のバランス、低精度演算支援、GPU 間直接接続性能の確保が設計上重要であるとの示唆が得られた。

3.2.4 アプリケーションからの Codesign フィードバック

本節では、コデザインに関する研究活動を概説する。アーキテクチャ部門は、各設計選択肢に複数の候補を盛り込んだシステムの基本設計の策定を行った。アプリケーション部門は、これらの設計選択肢に対してフィードバックを提供する役割を担った。

3.2.4.1 フィードバック収集手法

(1) Developer Survey (A/B) 質問項目

設計初期段階において、各 SubWG を通じて開発者アンケートを実施した。主な論点は以下である。

- CPU 上に残るカーネルの性質
- CPU 行列エンジンの有用性
- GPU メモリ容量要求
- ノード使用規模
- CPU-GPU コヒーレンシの必要性
- レイテンシ重要度

Survey B では、CPU残留カーネルのB/F比やノードメモリ要求をより具体的に調査した。

Developer Surveyの具体的な質問項目は3.2.4.4に示す。

(2) Fugaku ジョブ統計解析（FY2024）

より広範なアプリケーション特性を解析するために、スーパーコンピュータ富岳で実際に実行されたジョブを対象として統計解析を行った。2024年度を対象として、実行時間1時間以上のジョブを解析対象とした。解析対象ジョブ数は 3,129,958 であり、これらについて、ノード時間重み付き解析を実施し、以下を評価した。

- FLOPs 効率分布
- メモリ使用量分布
- B/F 比推定
- 通信量

3.2.4.2 主要設計項目に対する分析結果

(1) CPU設計

Developer Survey からは、以下のような意見が主流であった。

- CPU に残るカーネルは主に I/O、前後処理、分岐の多い処理
- 高演算密度カーネルは限定的
- 行列エンジン需要は明確でない

Fugaku実データからも、CPU側で高いFP効率を達成している例は少なく、ピーク性能の30%以上を達成したジョブは734件、50%以上は93件のみであり、全体としてメモリ帯域律速傾向が強いことが確認された。

(2) CPU-GPU接続

Developer Survey からは、以下のような意見が得られた。

- 高帯域を強く希望
- コヒーレントメモリは生産性向上に寄与
- メモリ容量選択と強く結合する設計項目である

(3) Scale-up / Scale-out ネットワーク

Developer Survey からは、以下のような意見が得られた。

- 典型利用規模は 25～64 ノード
- Pod サイズ 72 GPU で半数をカバー可能
- 帯域優先、レイテンシも一部重要

(4) GPUメモリ容量

Surveyでは128−256GBが中心帯域であり、512GB以上を希望するケースもあった。ただし、メモリ帯域律速性との整合を再評価する必要があると考えられる。容量要求は主に以下の要素に依存している。

- FFT スタック
- レプリカ・アンサンブル
- 大規模疎行列
- 粒子数

3.2.4.3 システムバージョン別フィードバック

本節では、アーキテクチャ部門から示されたハードウェア設計における選択肢と、それに対するアプリケーションからのフィードバックの内容について記述する。

- System v0.5
  - コデザイン項目 1：CPU 仕様
    - 関連する調査項目：Developer Survey (A) 質問項目 1, 2
      関連するジョブ統計：メモリ使用量の分布
    - 開発者調査からの主要な考察：
      - CPU 側の行列エンジンの利用方法については推測するに留まっている。具体的には、小型密行列演算、解析、代理 AI タスクなどが挙げられる。精度については、倍精度が有用との推測が示されたが、この結果については必ずしも根拠が明らかではなく、検証する必要がある。
      - 2 種類の CPU メモリ集類について、関連する質問は実施しなかった。
    - ジョブ統計からの主要な考察：
      トータルメモリ容量として、4x730 (=2,920) GB/ノードでジョブの 50%をカバーし、4x1600 (=6,400) GB で 90%をカバーする。
      GPU と CPU のメモリの両方に重複コピーが存在しないと仮定した場合、CPU 側メモリ容量については、
      - GPU が 128 GB/個、4 個/ノードの場合：2,408 GB でジョブの 50%を、5,888 GB で 90%をカバーする。
      - GPU が 256 GB/個、4 個/ノードの場合：1,896 GB でジョブの 50%を、5,376 GB で 90%をカバーする。

      SME の選択については、データからどちらが良いということはできない。GPU が非常に積極的な設計（すなわち FP4/FP6/FP8 のみ）であった場合、CPU SVE は優れたリスク調整因子となりえるため、GPU の仕様が明確であれば検討できることがあったかもしれない。

ジョブ統計からは、より大容量のメモリを支持すべきと考える。計算能力は常にメモリ容量より急速に成長する一方で、アプリケーション開発者は適応可能であるため、メモリ予測については慎重であるべきである。

- ✧ 推奨：
  - SME：A または B
  - メモリ：A

➢ コデザイン項目 2：CPU-GPU インターコネクト

- ✧ 関連する調査項目：Developer Survey (A) 質問項目 6
- ✧ 関連するジョブ統計：なし
- ✧ 開発者調査からの主要な考察：
  - 性能：可能な限り高い CPU–GPU 帯域幅と多数の同時転送が望ましい。開発者はプログラミングの容易性より性能を優先する。メモリ同期に伴うオーバーヘッドを懸念する声もあった。
  - キャッシュコヒーレンシ：生産性向上に寄与するが、オプションであり、ピーク帯域幅やユーザ制御を低下させないでほしい。調査対象の開発者の大半（ただし全員ではない）は、すでにコード内でデータ転送を明示的に管理している。少なくとも 1 つのアプリケーションの開発者は、「Unified Memory」向けに設計した際に、ハードウェアコヒーレント機能を持たない計算機での実行時の性能低下を懸念していた。

開発者はインターコネクトのより高い性能を求めており、したがって NVLINK の選択を支持する傾向にある。一部のアプリケーションは統合メモリを前提としたプログラミングを行っており、それをサポートしないプラットフォームでの性能低下を懸念している。

- ✧ 推奨：A

➢ コデザイン項目 3：スケールアップネットワーク

- ✧ 関連する調査項目：Developer Survey (A) 質問項目 5
- ✧ 関連するジョブ統計：富岳におけるノード使用率、B/F 通信量
- ✧ 開発者調査からの主要な考察：
  - 規模：約 25～64 ノード（100～256 GPU）
  - 中規模：100～300 ノード
  - 全体的傾向：多くのアプリケーションではポッド上限（≤576 GPU）まで、グランドチャレンジではフルシステム利用
- ✧ ジョブ統計からの主要な考察：
  - 72 GPU のポッドサイズでジョブの約 50%、576 GPU で約 90%をカバー見込み
  - スケールアップネットワーク帯域幅：0.4～4 TB/s/GPU（0.01～0.1 B/F × 200 PF × 20%）

ジョブ統計は、ポッドサイズ 72 を選択することを支持している。開発者はこれより大規模なサイズを目指す意欲があるため、階層型ネットワーク性能に対応したプログラミングを余儀なくされる。NVL72 の実ハードウェアが存在すれば、本設計項目について詳細な調査が可能である。

- ✧ 推奨：B3

➢ コデザイン項目 4：スケールアウトネットワーク

- ✧ 関連する調査項目：Developer Survey (A) 質問項目 7
- ✧ 関連するジョブ統計：B/F 通信量

✧ 開発者調査からの主要な考察：

- 帯域幅が最も重要である。
- レイテンシは allreduce/内積演算、3D FFT、および小メッセージにおいて重要であり、システムは多数の同時小サイズ送信によるスループット劣化を回避すべきである。

✧ ジョブ統計からの主要な考察：

- スケールアウトネットワーク帯域幅：600 GB/s/GPU（0.3 B/F × 10% × 200 PF × 20%）
- 20% → 計算効率の楽観的指標
- 10% → ポッド外通信比率の推定値

分析の結果、600 GB/s で十分であることが示唆される。ポッド外（長距離）通信においてどのような種類の通信が行われているかについて、より詳細な分析が必要である。

✧ 推奨：B

➢ コデザイン項目 6：GPU Memory

✧ 関連する調査項目：Developer Survey (A) 質問項目 3, 4

✧ 関連するジョブ統計：Fugaku Memory Use Data

✧ 開発者調査からの主要な考察：

- 実運用での多くの計算ケースをカバーする容量は、GPU あたり 128～256 GB である。
- 高容量：大規模配列、FFT スタック、レプリカ・アンサンブル、スーパーパーティクル、スパース直接解法などに求められる容量は 512 GB～1 TB である。
- 外れ値：グローバル LES や極めて大規模な MD／スパース問題では 1 TB 以上が必要であり、極限的な例では数 TB の搭載が求められる。
- 多数派の見解：ワーキングセットがデバイスメモリを超えると、バンド幅の増大により容量不足を補うことはできない。

✧ ジョブ統計からの主要な考察：

- メモリ容量：GPU あたり 730 GB で半数のジョブを、1600 GB で 90%のジョブをカバーする。

ジョブ統計および調査結果は、大量のメモリが必要であることを示唆している。しかし、この結論は、以下の理由により慎重に検討すべきである。第一に、この調査は総メモリ要件のみに限定されたものである。しかし、本質的なのは単一のメモリ帯域幅制約計算カーネルにおいて必要なメモリ量である。他の計算カーネルでのみ必要なデータは CPU メモリに格納可能であり、演算律速コードにおいては転送を隠蔽できる。さらに、調査ではユーザに性能を FP64 FLOPS で考えるよう求めた。これは演算律速コード（オーバーラップが効く場合）にのみ適用され、ユーザは得られる性能を過大評価する可能性がある。もし期待されるメモリ帯域幅で聞いた場合は、ユーザは問題サイズを縮小すると考えられる。

✧ 推奨：A

➢ コデザイン項目 7：CPU SVE Performance

✧ 関連する調査項目：Developer Survey (A) 質問項目 1

✧ 関連するジョブ統計：なし

✧ 開発者調査からの主要な考察：

- CPU カーネルは引き続き多くのコードで使用される。 主な用途としては、I/O や圧縮・展開処理、前処理・後処理、MPI の制御やコレクティブ通信、並列度の低いカーネルや分岐の多い処理、四倍精度演算などが挙げられる。

・ 複数のアプリケーションで CPU と GPU の並行使用が計画されている（例：粒子–流体の分割処理、CPU での解析・AI と GPU メインループの並行実行）。一方、クロスデバイス共有を避けるため、大規模環境ではすべての処理を GPU 上で完結させるアプローチをとるアプリケーションもある。

AI の実行に CPU 上の SME を活用するというアイデアをアンケートで質問した。同様に、メモリ制約により GPU のプログラミングが困難な場合に CPU を活用する可能性についても設問に加えることができるだろう。富岳のジョブ統計についてはフォローアップを行い、SVE の現在の活用状況を裏付けるデータが得られるか確認する必要がある。また、各カーネルについて、SVE で対応可能かどうか、また対応可能な場合は演算律速であるかどうかを把握するため、追加の設問を設ける必要がある。

✧ 推奨：B

- System v1.0
  - ➢ コデザイン項目 1a：CPU 仕様 – 行列演算エンジン

手法：CPU に残る可能性が高いカーネルを調査した。それらの B/F 値はどの程度か。行列演算エンジンを活用できるか。標準的な演算でコアを飽和させることができるか、またはコア数を削減してもデメリットはないか。AI に向けたオプション機能の有用性を調査した。エミュレーション性能の見通しを調査した。

✧ CPU に残るカーネルと B/F 値についてアンケートで調査（Developer Survey (B) 質問項目 1）。

✧ 行列演算エンジンについて改めてアンケートで確認（Developer Survey (B) 質問項目 2）。

✧ オプション機能の有用性について AI チームに調査を依頼。

✧ エミュレーション性能について数値計算ライブラリチームに調査を依頼。

調査からの主要な考察：

CPU 行列演算エンジンの活用方法について、開発者たちは依然として具体的な見通しを立てられずにいる。その主な理由は、対象アプリケーションの多くが密行列積を主要なカーネルとして使用していないことにある。CPU に残るカーネルの例としていくつか挙げられたものの中にはベクトル化が可能なものもあったが、演算負荷の高いものは少なかった。また、ネットワークに近い場所にデータを置く必要がある場合に CPU を活用するという考え方も示されたが、これはスケールアップネットワークの設計に依存する。

AI チームに調査の主要な考察：

AI チームは、オプション機能の必要性を把握するため、深層学習アプリケーションで使用される数値精度について調査を行った。彼らは、HPC 志向タスクの一部では FP32/FP64 が主力として使用される精度であり、トレーニングおよび推論タスクのための予備手段として FP16/BF16 が必要であることを見出した。また、トレーニングと推論の両方で FP8 が主な標準であること、INT8 が既存の互換性とベンダ間の互換性を保つために不可欠であること、そして FP6 および FP4 は将来的に有望な可能性があることも発見した。学習においては、より高い精度での累算が不可欠である。トランスフォーマー演算では混合精度も活用されている。全体として、F16F16 や B1616 といったオプションが有用であるという根拠はほとんど見られなかった。I64I64 についてもユースケースは確認されなかった。一方、FP6 または FP4 のオプション機能については、大事なユースケースが見出された。

計算ライブラリチームに調査の主要な考察：

FUJITSU-MONAKA-X における浮動小数点演算のエミュレーションのパフォーマンスモデルを、数値ライブラリグループが用意した。単精度および倍精度のエミュレーションには、Ozaki 方式 2 を使用して Int8 を使うことが勧められている。16384 サイズの行列では、FP64 をエミュレートする方がオプショナル機能 A のネイティブ仕様を使うよりも良い。2048 サイズの行列では、オプション A のネイティブピーク性能に比べ約 20%のパフォーマンス低下が予想される。

✧ 推奨：B

- コデザイン項目 1b：CPU 仕様 – メモリ

手法：ノードあたりのパフォーマンスを考慮して、CPU と GPU の合計メモリ要件をモデル化し、CPU 側で実行されるカーネルが使用するメモリ量とバンド幅（B/F）を確認した。

✧ 合計メモリ要件に関する調査 (Developer Survey (B) 質問項目 3)。

✧ CPU カーネルの特定のメモリ要件に関する調査 (Developer Survey (B) 質問項目 4)。

開発者調査からの主要な考察：

多くのアプリケーションにおいて、CPU 側のメモリ要件は小さかった。なぜなら、開発者はほとんどすべてを GPU に移行することを計画していたからである。しかし、一部の例外があり、大幅に異なるサイズのメモリ要求が出ていた（その中には非常に大きな容量も含まれた）。開発者は依然として GPU 上で多くのメモリが必要であり、これが総システムメモリの大きな要求につながっている。データをネットワークに送る前に CPU 上でステージングする必要がある場合、より多くの CPU メモリが必要になる可能性について懸念が上がった。

GPU のメモリ容量が減少した場合、より大きなメモリを持つ CPU は重要性を増す可能性がある。開発者は CPU 計算カーネルにメモリ帯域幅制限を課すことを示唆していたが、これらの計算カーネルが主なコストとなることは考えられていなかった。これらの設計選択肢は完全には固定されておらず、将来的にメモリ容量の追加が容易になることから、A は安価な選択肢となり得る。

✧ 推奨：A

- コデザイン項目 *2*：CPU-GPU 相互接続

手法：未最適化のアプリケーションを用いて評価した場合、誤解を招く可能性がある。代わりに、NVIDIA に最適化されたアプリケーションに基づいた証拠を提供してもらうよう依頼した。NVIDIA は、NVLINK の利点をリストし、パフォーマンス差異に関する情報を共有した。

NVIDIA のレポートから得られる主なポイント：

NVIDIA は、NVLINK が性能上の利点をもたらすアプリケーションを強調したプレゼンテーションを示した。その例にはシミュレーションアプリケーション、ディープラーニング、HPL ベンチマークが含まれていた。ただし、CPU と GPU の性能を同時にモデル化する複雑さにより、明示的な両者の選択肢の比較は提供されなかった。いくつかのアプリケーションでは、GPU が提供するメモリ以上の容量が必要であり、その場合は高速インターコネクトによりデータを CPU 上で保持できるようになった。また、キャッシュコヒーレントとマネージドメモリ間のパフォーマンス差を示す例も提供された（NEMO）。議論の中で、NVIDIA は、キャッシュコヒーレンシの利点を抽出することは困難であるが、主にファイングラニュラリタスクにとって有益であると主張した。また、高速インターコネクトと統一メモリの概念モデルにより、アプリケーションの移植初期段階でスピードアップを容易に実現できることも示唆した。

EEA が GPU 最適化されていないため、2 つの選択肢のパフォーマンス差をモデル化することは困難である。NVIDIA と RIKEN の両方において、インターコネクト速度の異なるオプションに対するパフォーマンスシミュレーションの方法論が欠如しているようである。そのため、特定のアプリケーション要件に基づいてコードザイン上の決定を行うことはできない。

開発者はアンケートで高速インターコネクトを望んでいる。NVIDIA のデータによると、十分に最適化されたアプリでも、様々なタスクでは依然として高速インターコネクトが必要であることが示唆されている。高速インターコネクトの利点は、GPU メモリに関する選択肢と結びついている。これは、将来小容量の GPU メモリを

選択した場合に備えての手段となる。回答者の大部分はデータ転送を自分で管理したいと答えている（そして細かい制御を望んでいる）が、少なくとも一部（SALMON 開発者）は統合モデルを使用しており、コヒーレントメモリが利用できない場合のパフォーマンスとコード品質の低下を心配している。

✧ 推奨：A

➢ コデザイン項目 7：CPU の SVE パフォーマンス

手法：CPU 上で実行されるカーネルを特定し、それらに焦点を当てて SVE パフォーマンスの必要性を理解する。「Vector Pipeline Business」またはピークベクトル FLOPS に対する FLOPS 利用率を調べる。AFX64 よりも潜在的に長い待ち時間に注意する。

✧ CPU 計算カーネルの調査（Developer Survey (B) 質問項目 1）。

✧ FLOPS 使用率のジョブ統計。

ジョブ統計からの主要な考察：

本調査の方法は、富岳における現在の SVE 効率を調べることにある。もし富岳でピーク FLOPs が達成されることがほとんどない場合、それは FugakuNEXT の CPU 上で残っているカーネルも浮動小数点演算によってボトルネックになるとは考えにくいことを示唆している。

前年度の富岳ジョブミックスデータを分析し、実行時間が 1 時間を超えるすべてのジョブを調査した。合計で 3,129,958 件のジョブを対象とした。データから FP 性能とピーク FP 性能の比率に関する分布を作成した。その結果、ほとんどのジョブがピークの 20 パーセント未満で動作していることが明らかになった。ピークの 30 パーセント以上を超えたのはわずかに 734 件、ピークの 50 パーセント以上を超えたのは 93 件のみであった。

富士通からの提案を受け、富岳の FS2020 で測定されたアプリケーションが、メモリバスの活用率が 60 パーセント未満でありながら FP 効率が 30 パーセント以上であるかどうかをさらに調べた。しかし、この両方の基準を満たすアプリケーションは存在しなかった。

✧ GENESIS: FP 効率約 4 パーセント、メモリバス使用率約 5 パーセント。

✧ SCALE: FP 効率約 3.71 パーセント、メモリバス使用率約 9.22 パーセント。

✧ LETKF: FP 効率約 0.07 パーセント、メモリバス使用率約 0.10 パーセント。

✧ LQCD-DWF-HMC: FP 効率約 12.5 パーセント、メモリバス使用率約 67 パーセント。

✧ FFB: FP 効率約 11.51 パーセント、メモリバス使用率約 43.09 パーセント。

✧ EbE-method: FP 効率約 6.20 パーセント、メモリバス使用率約 14.53 パーセント。

✧ SALMON (基底状態計算): FP 効率約 3.94 パーセント、メモリバス使用率約 9.77 パーセント 。

✧ SALMON (実時間進化): FP 効率約 7.32 パーセント、メモリバス使用率約 8.11 パーセント 。

富岳のジョブデータから、コード全体はほとんどメモリバンド幅によってボトルネックになっていることがわかる。開発者調査では、CPU に残ることを想定している計算カーネルは、それほど高い演算強度を持っていない。したがって、2-flow が最適な選択肢であると考えられる。

✧ 推奨：A

● その他のフィードバック

➢ FUJITSU-MONAKA-X CPU の 512 ビット幅 SVE ベクトルの重要性を検討した。GPU への移植活動が進行中であるため、将来のコードの CPU ボトルネックになる可能性のある計算カーネル群を予測するのは難しい。その代わりに、一部の EEA コードを実行し、その全体的な動作を分析してみた。富岳で実行した場合、-Ksimd_reg_size=256 オプションを使用して SVE サイズを 512 ビットから 256 ビッ

トに手動で変更できた。また、中間プログラムを作成して正しい環境設定を行う必要があった。いくつかのアプリケーションにおいて、512 ビット SVE を使用することで 256 ビット SVE に対して計算速度の向上がみられた。しかしながら、この結果は、SVE サイズだけでなく、コンパイラの最適化等の影響も含まれており、SIMD 幅の影響を正確に見積もることはできていないことに注意が必要である。

3.2.4.4 Developer Survey 質問項目

Developer Survey (A) 質問項目

1. CPU設計: FugakuNEXTのCPUに期待することは何ですか？FugakuNEXTにおいてあなたのアプリケーションではCPU上で実行するカーネルが残りそうですか？また、あなたの分野で、CPUとGPUを同時に活用しているコードの実例がありますか？
2. 一つの可能性として、CPUに行列計算エンジンを追加することが検討されています。一部のアプリケーションでは、GPU上で重い計算カーネルを実行し、CPUでAIを動かすことでデータのスワップを避けるなどにより、このエンジンが有効であることが示されています。あなたのアプリケーションにおいて、CPU行列計算エンジンを活用するアイデアはありますか？もしある場合、行列エンジンが対応すべき数値精度（例えば、半精度、単精度、倍精度）について要望がありますか？
3. 各GPUがFP64で約200 TFLOPS、FP16で10 PFLOPSの性能を持つシステムを想定してください。そのGPUであなたのアプリケーションを実行するためにはどれくらいのメモリが必要になりそうですか？具体的数字として、128GB、256GB、512GB、768GB、1TBといった値を教えてください。ここで、CPU側には大量のメモリがあると仮定してください。加えて、必要メモリ容量を規定する制約（例えば、GPU間で分散が難しい大規模データ構造があるなど）があれば教えてください。
4. 上の質問に関連し、メモリ容量と転送速度のトレードオフについてお聞きします。GPUメモリについて、メモリサイズが減る代わりに大幅に帯域が増える可能性があるとします。もしあなたのアプリケーションがメモリ容量に関する制約が厳しい場合、帯域の大幅な増加によってメモリ容量に関する問題を回避できそうですか?
5. 再度、質問3のGPU性能を想定します。FugakuNEXTにこのGPUが13,600基あり、3,400ノード（1ノードあたり4 GPU）に分散されていると仮定します。FugakuNEXTでは、4から576 個のGPUで“pod”と呼ばれるグループを構成し、pod内の通信は、podを跨ぐスケールアウトネットワークよりも高速なスケールアップネットワークで結合する構成が検討されています。pod のサイズを決める際に、実用的なアプリケーションにおいて、典型的に使用するノード数が重要になります。あなたのアプリケーションでは、典型的に使用するノード数はどれくらいになると想定されますか？
6. CPUとGPU間の結合について、NVLINK Fusion や PCI Express が検討されています。NVLINK の利点は、CPUとGPU間でメモリがコヒーレントであることです。この機能はあなたのアプリケーションの開発の効率性を大幅に向上させると考えられますか？
7. システムのネットワークについて、バンド幅とレイテンシの両方がアプリケーション性能に影響する可能性があります。あなたのアプリケーションにおいてレイテンシは重要ですか？
8. HPCにおいて、コードによってメモリ帯域幅やキャッシュサイズ、計算時間など制約される要素が異なります。あなたの科学分野において、メモリ帯域幅制約ではないアプリケーションがあれば教えてください？
9. あなたの分野のHPCシミュレーションアプリケーションにおいて、AIタスクが最も計算コストの高い部分となるものはありますか？

Developer Survey (B) 質問項目

1. 前回のアンケートでは、CPU に対する期待について伺いました。今回は、CPU に残るであろう計算カーネルのなかで計算コストの高いものについてお尋ねします。これらのカーネルのおおよその byte / flop 比はいくつくらいでしょうか。また、これらのカーネルのベクトル化は可能ですか。
2. 前回、CPU 上の行列エンジンをどのように活用できるかについてアイデアがあるかを伺いました。もしそれ以降に新しいアイデアが思いついたなどということがあれば共有していただけますでしょうか。前回の質問では、GPU でシミュレーションを行い、CPU 側で AI を動かすという可能性を 1 例として示しました。もう一つの可能性としては、行列演算を含むカーネルがあるが、その関連データを GPU の小さいメモリに移動させたくない、といった場合が考えられると思います。
3. 前回のアンケートでは、GPU あたりどの程度のメモリが必要かを伺いました。今回は、ノード全体（CPU + GPU）でどの程度のメモリを必要とするかを伺いたいです。1 ノードに GPU が 4 基あると仮定してください。前回は、1 GPU が 200 TFLOPS の性能を持つと仮定した上で見積もっていただきました。しかし、コードが FP64 計算性能ではなく、メモリ帯域幅で律速されている可能性もあります。そのため今回の想定では、GPU あたりのメモリ帯域幅が 50 TB/s と仮定してください。
4. CPU メモリは次の 2 つの用途に使えると考えています。
   1. 後で GPU に転送して処理するデータを格納する
   2. CPU 側で処理するデータを格納する
5. CPU に残ると想定される計算コストの高いカーネルを考えてください。これらが処理するデータはどれくらいのメモリ容量を必要としますか (上の 2 つ目の用途)。

3.2.5 早期アプリケーション評価結果

基本設計開発報告書において NVIDIA 社から報告されたアプリケーション推定性能をもとに、対富岳性能比を算出した。なお、性能比の計算に用いた富岳における性能については、基本設計において理研で測定されたものである。6 本のEEA1のうち、E-Wave については、基本設計においては富岳における性能測定を行っていないため、残りの 5 本のアプリケーションについての値を以下に示している。

NVIDIA 社の推定性能では、富岳NEXT のハードウェアとして、メモリ構成 3 種類と、ネットワーク構成 12 種類を想定している。ネットワーク構成の影響評価は、LQCD-DWF-NMC のみで行われており、メモリ構成の影響評価はすべてのアプリケーションで行っている。

3.2.6 富岳 NEXT 時代にむけた 新しいアプリケーションの開発

3.2.6.1 AI アプリケーション

FugakuNEXTは人工知能研究のための最先端のコンピュータとなることが期待されている。最新のAIアクセラレータを基盤とした設計により、富岳と比較してAI学習および推論タスクにおいて大幅な高速化が実現されることが予想される。AI計算能力の成長率は従来のFP64演算よりも著しく高くなることが予測されており、シミュレーションアプリケーション開発者にとってAI手法を統合することは革命的な進歩を実現するために不可欠となる。アプリケーション分野の目標は、FugakuNEXTにおける将来のAI利用事例を予測し、そのためのシステム準備を支援することである。

EEAアプリケーションの大部分は、計算ボトルネックとしてAIを組み込んでいない。HPCとは異なり、AI分野では高度なカスタムコードではなく標準ライブラリに頼る傾向にある。また、この分野は急速に変化しているため、シミュレーションコードと同様の方法で特定のAIのEEAアプリケーションを直接選択することは現実的ではない。代わりに、私たちは新たなパターンがどのように登場し、システム設計にどのような影響を与えるかを理解しようとした。また、NVIDIAは商用AI用途（LLM推論など）のチューニングに内部的に注力していると予想されるため、当初の焦点をAI4Science用途およびAI-HPC融合に絞るべきだと考えた。

コデザインの要点

これらは、AIアプリケーションに関連するコデザインの重要な点である：

- GPU メモリ容量
- スケールアップネットワークの規模
- スケールアウトネットワークの性能
- FP64 ベクトル演算性能と AI 演算性能の比率
- システムの非一様性
- 専用ノードの必要性

同一ノード上へのAI・HPCの混在配置（コロケーション）と、AI用・HPC用に独立したパーティションを確保しネットワーク経由で結合する分離配置とは、いずれも重要なシステム構成パターンであり、両者を等しく検討の対象に含める必要がある。パーティションが分けられている場合、ハードウェアの弱点をソフトウェアで補えることができるが、カップリングが強い場合はハードウェア側に注意を払う必要がある。スケールアップネットワークサイズとGPUメモリ量のカップリングについても検討する必要がある。また、「AI as a Service」において、ユーザがこのサービスと特化した方法でインタラクションしたり、特化したノードで実行したりすることで得られるメリットについても考慮すべき点だ。これはユーザーインタラクション戦略に影響が考えられる（例：SLURMの代わりにKubernetesを使う）。

計算パターン一覧

HPCアプリケーションにおける推論呼び出し：

- AI ライブラリと HPC コード間の相互運用性を確保する上での主要な課題は、オーバーヘッドなしに実現することである。
- コデザインの要点：コロケーション時には、十分な GPU メモリ量を確保する必要がある。FP64 ベクトル性能と AI 行列性能のバランスが重要である。
- コデザインの要点：専用パーティションを与えられた場合には、遅延が低いスケールアウトネットワークが必要である。

HPCアプリケーションによる学習の呼び出し：

- HPC およびシミュレーションを個別のプロセスとして実行するよりも早く学習を終了させる。
- コデザインの要点：この操作はスループットが重視されるものであり、レイテンシよりも優先される。専用パーティションを使う場合、非常に高いスケールアウト帯域幅が必要である。

推論によるHPCアプリケーションを呼び出す：

- このパターンは同期動作であるため、型結合が必要となる可能性が高い。
- コデザインの要点：GPU メモリ量、FP64 ベクトル性能と AI 行列性能のバランス。

学習によるHPCアプリケーションを呼び出す：

- 低遅延を実現するためには、学習エポックが長時間かからないように、高効率なカップリングが必要である。
- コデザインの要点：GPU メモリ量、FP64 ベクトル性能と AI 行列性能のバランス。

AI推論機能を提供するサービス：

- モデルカタログまたは独自のモデルを提供する。バッチスケジューリングとは異なる方法でインタラクションを行う。
- 注意点：ユーザが大規模なモデルカタログにアクセスする場合、モデル切り替えのコストは許容できる範囲内にあるか？
- コデザインの要点：システムの非一様性、GPU メモリ量、スケールアップネットワークの規模。

AI学習機能を提供するサービス：

- 小規模モデルでのハイパーパラメータ最適化は重要であり、強スケーリングが必要である。
- コデザインの要点：システムの非一様性、GPU メモリ量、スケールアップネットワークの規模。

AIモデルの微調整機能を提供するサービス：

- コデザインの要点：システムの非一様性、GPU メモリ量、スケールアップネットワークの規模。

AIコーディング：

- システムソフトウェアやリソース管理においては、コデザインよりも重要である。
- コデザインの要点：何千ものエージェントがコードを並列処理するために並列に展開されることを想定すると、高速な CPU が最も重要となる。

エージェントAIと科学ワークフローの統合：

- 実にどのような形で実現するかはまだ明確ではない。
- コデザインの要点：ここでは、AI タスクの指揮者として CPU が非常に重要となる可能性がある。

「End-to-end」AI推論：

- ユーザが AI 推論サービスパターンを使用しないシナリオはあるか？
- LLM と AI4Science モデルの間には何が異なるのか。モデル規模、コンテキストウィンドウのサイズ、プロンプトキャッシュの可否、…
- コデザインの要点： GPU メモリ量、スケールアップネットワークの規模。

「End-to-End」AI学習

- ユーザが、サービスとして提供される学習ではなく、独自に構築したカスタム環境でモデルを作ることを望む場面が生じうるか。
- コデザインの要点： 基盤モデルを作るために、マシンの全体を使用する？ 大規模言語モデル構築と比較して、この作業には何が異なるのか？

今後の課題

今年度に、FugakuNEXT上でのAI利用パターンを調査し、それが設計上の決定にどのように影響するか検討した。将来的なコデザインフィードバックとして、これらのユースケースをそれぞれ検討し、トレードオフを考慮する必要がある。これらの取り組みは、ベンチマークセットに直接統合できるAIアプリケーションを探し出すことにも重点を置く。具体的には、LAMMPSやGROMACSによるMILP、NICAMによる気象予測、ビジョントランスフォーマーによる運用中のイベント検出などである。これらの取り組みについては、より広範なコミュニティとの間でSubWGを通じて継続的に議論を行い、新興するAIアプリケーションやユースケースを予測する。

3.2.6.2 低精度演算器の活用

深層学習を実行するための計算資源に対する需要の高まりは、ハードウェア分野に影響を及ぼしている。近年のアクセラレータは、深層学習の演算に適した専用の行列演算エンジン（Tensorコアなど）を搭載しており、これらは従来のHPC向けハードウェアとは大きく異なる特徴を持つ。特に、低精度（FP16、FP8、INT8）での演算において極めて高い効率を示す。ハードウェア会社は、計算科学で広く用いられてきたFP64/FP32精度よりも、これら低精度演算を優先的に最適化するようになっている。したがって、これらの新技術を活用することが、FugakuNEXTにおいて飛躍的な性能向上を実現するための重要な戦略となる。

GPU Model	FP64 (ベクター)	FP64 (行列)	FP16 (行列)	INT8 (行列)
A100 SXM	9.7 TFLOPS	19.5 TFLOPS	312 TFLOPS	624 TOPS
H100 SXM	34 TFLOPS	67 TFLOPS	990 TFLOPS	1979 TOPS
B200 HGX	37 TFLOPS	37 TFLOPS	2250 TFLOPS	4500 TOPS

表：各世代のハイエンド GPU のパフォーマンス比較。行列演算の処理速度を、スパース性は考慮せずに示す。

低精度ハードウェアの開発は多方面にわたる影響をもたらす。まず、行列乗算に帰着できるアプリケーションに対しては、有限精度での計算を克服するための戦略を策定することが重要になる。演算律速（compute-bound）ものの、密な線形代数としては扱えないコードについては、ハードウェアとアルゴリズムのコデザインが求められる。演算律速でない手法に対しては、低精度数値手法の進展によりメモリ帯域幅や通信コストを削減するアプローチが提供される。これらすべての研究は、低精度データ型を効率的に扱える高度なプログラミングモデルの整備によって支えられる。

低精度演算とコデザイン

密な線形代数に基づくアプリケーションに、最も大きな機会がある。開発者は、ずっと前からアルゴリズムを行列積の形で記述すれば、ハードウェアから最高性能を引き出せることを認識してきた。したがって、行列積がボトルネックとなっている最先端の HPC コード例も少なくない。SubWG2 の研究が示すように、特に物理・化学分野におけるテンソル収縮ベースの手法でも、行列積中心の手法に対する機会が存在する。これらの手法は、近代ハードウェア上でコストが削減されるため、今後ますます普及するであろう。低精度による誤差は、Ozaki スキーム、Markidis 法、あるいは BF16×9 といったエミュレーションアルゴリズムを用いることで克服できる。しかし、FP4 や FP6 といった極めて低精度の演算しか利用できない場合に、エミュレーションが実用的に効率的であるかどうかは未解決の課題である。開発者への相談から浮かび上がったもう一つの課題は、多数の小さな行列積を並列処理するためのエミュレーション技術を確立することが重要である。

行列積行列積に依存する計算律速アプリケーションの別クラスとして、深層学習が挙げられる。 低精度戦略は

深層学習分野で広く認識されているが、BF16、FP8、INT8 などの精度への志向は主に大規模言語モデルの開発から促進された。計算科学においては、未だに多くのモデルが FP64 または FP32 を基盤としている。例えば、SubWG2 は機械学習に基づく原子間ポテンシャルの精度要件を調査し、FP32 や TF32 が用いられているものの、他の精度設定はほとんど試みられていないことが判明した。科学分野の AI モデルは比較的小規模であり、精密な数値予測を必要とするため、FP32 や FP64 のような高精度浮動小数点演算が求められている。コデザインの観点からは、これらのモデルが効率的動作することを保証するために GPU のコデザインが不可欠である。アプリケーション側では、浮動小数点エミュレーションの検証および低精度型を利用可能とするアルゴリズムの探索が重要になる。

多くのコードは、Tensorコアの利用は容易ではない。演算律速アプリケーションすべてが密な線形代数として帰着できるわけではない。EEAの中で、GENESIS 分子動力学コードがその一例である。これらのコードは、特定の問題設定下では演算の一部でTensorコアを活用できるかもしれない（例として、フーリエ変換の代わりに高速多極子法を用いる場合など）。しかし、これが汎用的な解決策となるわけではない。FugakuNEXT 開発チームは、これらのアプリケーションを適切に支援するために、FP64／FP32 ベクトル演算性能と AI 行列演算性能の比率を慎重に決定することが重要である。

EEAの大半は、メモリ帯域幅律速なアプリケーションである。さらに、いくつかはネットワーク遅延に対して極めて感度が高いこともある。ずっと前から、メモリ性能の向上は計算性能の伸びに比して緩やかであった。したがって、FP64／FP32 のベクトル演算性能が大幅に低下しても、メモリ帯域幅の増大によって全体のスループットが向上するケースが多い。目標は、メモリおよびネットワーク性能を向上させるため、より低精度の数値形式（FP16、BF16等）を使用することである。 従来、ほとんどの GPU はネイティブに FP32 のみを提供しており、現在でもコンシューマ向けハードウェアは FP64 性能が限定的である。こうした先行研究に基づき、アプリケーション開発者は精度を削減した手法を採用できる（分子動力学分野で成功例が報告されている）。AI 技術の進展に伴い、低精度型をサポートするプログラミング言語の拡充が進んだ。 併せて、コード中の精度要件を自動的にプロファイルする新たなツール（例：R-CCS の RAPTOR）も登場している。

今後の課題

FP64／FP32 ベクトル演算性能と AI 低精度行列演算性能との比率は、次年度に計画されているコードコデザインの重要項目である。この項目へのフィードバックのために、SubWG メンバーおよびアプリケーション開発者に対し、演算律速アプリケーションの実態と、他の研究グループが低精度演算の活用にどのように取り組んでいるかを把握する調査を実施する。数値ライブラリチームと協働し、密な線形代数を利用するコードを用いて、浮動小数点エミュレーションを適用した際の性能向上を予測する。併せて、解析対象とする AI アプリケーションの代表的使用例を抽出する。さらに、行列演算を使用しない演算律速アプリケーションを選定し、FP64 のサポートが限定的なハードウェア上で実験的に最小限の浮動小数点要件を評価する。プロジェクトの残りの期間においては、アプリケーション開発者と連携し、演算律速またはメモリ帯域幅律速計算カーネルが低精度を活用できる使用例を特定し、チューニングを支援する。これらの使用例に適用した技術とノウハウは研究コミュニティへ共有し、計算科学者が FugakuNEXT の恩恵を最大限に活用できるよう備える。

3.2.6.3 量子計算

量子古典ハイブリッド計算は、符号問題を代表とする古典計算では困難な問題を量子計算を交えて解決する手法として近年端緒が開かれた。この流れは富岳NEXT時代にも継続、さらには発展する可能性があり、評価対象アプリケーションのカテゴリに加える必要がある。来年度、対象アプリケーションの検討追加を進めるためのグループ

を 既存のアプリケーションSubWGに追加、もしくは新 SubWGに構築することを検討する。

3.2.7 アプリケーション開発における対外連携

3.2.7.1 「富岳 NEXT」が切り拓く未来に関するヒアリング

HPCの活用分野を代表される方々を訪問し、富岳NEXTが切り拓く2030年代の科学と社会についてヒアリングを継続している。ハード・アーキテクチャへの提言、AIの積極的なシミュレーションへの活用、オープンスタンダードへの対応促進、社会・産業インパクトを先取りするリアルワールド活用、GPU移植・チューニングを支える人材育成、持続可能なエコシステムの形成、などの幅広い知見を頂き、富岳NEXTの開発に活かしている。

分野	所属機関、氏名
生命科学	理化学研究所　BDR　マルチモーダル AI 基盤技術研究チーム　チームディレクター　小島 諒介
新物質・エネルギー	東京大学大学院　理学研究科　教授　常行 真司
気象・気候	国立環境研究所　環境情報部（企画・支援部門）　研究情報室　室長代行　八代 尚
地震・津波防災	東京大学　地震研究所　巨大地震津波災害予測研究センター　教授　市村 強 海洋研究開発機構　海域地震火山部門　地震津波予測研究開発センター　センター長　堀 高峰
ものづくり	日本大学　研究特命教授　加藤 千幸
	神戸大学　教授　坪倉 誠
	東北大学　教授　河合 宗司
基礎科学	高エネルギー加速器研究機構　素粒子原子核研究所　理論センター長・教授　橋本 省二
	筑波大学　教授　大須 賀健
デジタルツイン	大阪大学大学院　情報科学研究科　情報ネットワーク学専攻　教授　山口 弘純

3.2.7.2 「富岳 NEXT」アプリケーションセミナーの主催

富岳の後継機として開発が推進されている「富岳NEXT」は、我が国の科学技術・産業競争力を支える次世代の計算基盤として大きな期待が寄せられている。その実現に向けて、HPCアプリケーションのさらなる高度化と、それを担う人材の育成が不可欠である。こうした背景のもと、次世代計算基盤アプリケーション開発ユニットでは、「富岳NEXT」時代を見据えたHPCアプリケーション開発に関するAI活用も含めた最新動向や実践的な知見を広く共有するセミナーを企画する。

2025年度は下記のセミナーを実施し、のべ約1000名の参加者となった。

回数	日程	講師(敬称略)	タイトル	参加人数
1	5 月 28 日	青木 保道	アプリケーション開発エリアキックオフ	-
2	7 月 17 日	中村 宜文	Beyond CI/CD: Continuous Performance-Driven Principles for Future HPC Systems	65

			and Applications	
3	7 月 31 日	Mohamed Wahib	Vision AI for Science and Engineering Applications	76
4	8 月 20 日	尾崎 克久	Ozaki Scheme I: Emulation method for Matrix Multiplication	72
5	8 月 27 日	William Dawson	Applying the Ozaki-scheme to HPC Applications	70
6	9 月 25 日	椋木 大地	Challenges and Prospects in Automatic Generation of HPC Codes Using Generative AI	95
7	10 月 2 日	村井 均	Programming Environment for FugakuNEXT	123
8	10 月 21 日	下川辺 隆史	Miyabi: JCAHPC's New GPU-Based Supercomputer and Application Porting Activities to GPU	65
9	10 月 30 日	藤田 航平	Simultaneous use of CPU and GPU for fast and energy-efficient implicit wave simulation	77
10	11 月 10 日	吉田 亮	Integration of Simulation and the Real World through Sim2Real Machine Learning: Toward the Discovery of New Materials	85
11	12 月 2 日	Andreas Herten	Building JUPITER: From Procurement to Deployment	67
12	1 月 21 日	小島 諒介	Development of multimodal AI frameworks and foundation models in life sciences	61
13	2 月 18 日	平島 敬也	Star-by-star simulations of the Milky Way accelerated by AI surrogate modeling	60
14	2 月 26 日	白川 知功	Quantum-HPC Application Development for Quantum Many-Body Systems: Selected-Basis Diagonalization and Tensor-Network Approaches	64

また、広報部門と協力し、理化学研究所 計算科学研究センターの公式Webページで情報を発信している。

日本語　　: https://www.r-ccs.riken.jp/fugaku-next/app-seminar/

英語　　: https://www.r-ccs.riken.jp/en/fugaku-next/app-seminar/

2026年度も同様にセミナーを開催予定であるが、テーマを決め、そのテーマを深堀するようなセミナーとする予定である。また、本セミナーは一般財団法人 高度情報科学技術研究機構 神戸センターからの協賛を得て、HPCIアカウント保有者に対してメーリングリストでの告知を実施していただいた。2026年度は更に広くセミナーを告知し、富岳NEXTの認知度の向上とアプリケーション開発力の向上に資する活動とする。

3.2.7.3 国内連携

当プロジェクトのアプケーションエリアの活動において、富岳NEXTの潜在的ユーザアプリケーション開発者との連携が重要である。開発体制で述べたように、アプリケーションワーキングループは国内HPCアプリケーションコミュニティとの連携体制を組んでおり、その活動において初期評価対象アプリケーションを設定した。年度後半では、RISTの新センターとしてHAIRDESC(朴泰祐 センター長)が発足し、文科省プログラム-現行の成果創出加速課題の後継となるグランドリーチプログラムが2026年度開始として公募開始となり、HAIRDESCが課題のアプリケーションGPU化のサポート／伴走を行うこととされた。理化学研究所はHAIRDESCの協力機関であるだけで無く、グランドリーチプログラムから富岳NEXT利用を目指すアプリケーションのコデザインへの取り込みを目標として、より緊密な連携を行うべく定期検討会を行っている。

3.2.7.4 国際連携

アプリケーション開発における国際連携は、富岳NEXTを中心としたアプリケーションエコシステムの国際的展開、また、国際的に利用実績、需要のあるエコシステムが高効率で活用できるシステム構築の観点で重要である。今年度その一環として、EEA開発者を対象としたCI/CBフレームワークである Benchpark の活用ハッカソンを Lawrence Livermore National Laboratory (LLNL) のBenchpark開発者と協働で、理化学研究所・計算科学研究センターにおいて開催した。今後もLLNLに留まらず、米国DOE関連研究所をはじめ、国際連携を進めていくことにより、富岳NEXT開発のみならず、富岳NEXTユーザの利となる環境構築を進める。

3.3システムソフトウェア

3.3.1 プログラミング環境

3.3.1.1 全体システム・CPU 部

3.3.1.1.1 コンパイラ・プログラミング言語

3.3.1.1.1.1 プログラミング言語

HPC分野のアプリケーションでは、プログラミング言語として主にC、C++、Fortranが使用される。性能を出すにはそれらのコンパイラが重要である。「富岳NEXT」では、これらのプログラミング言語をサポートするコンパイラを提供する。CPU部のコンパイラでは、Arm C Language Extensions (ACLE)やOpenMPといったプログラミング言語の拡張仕様にも対応する。サポートする言語規格の版数や拡張仕様は、後述するコンパイラに依存する。

AI分野のアプリケーションでは、それらに加え、主にPythonが使用される。性能を出すにはPython用の数値計算などのライブラリが重要である。Pythonインタープリタとそのライブラリについては、ほかのWGと連携して詳細設計で検討する。

3.3.1.1.2 コンパイラ

3.3.1.1.2.1 CPU 部のコンパイラ

「富岳」では、計算ノードの演算装置がCPU (A64FX)のみであった。そのA64FXの機能・性能を最大限に発揮させるために、オープンソースや商用の既存のコンパイラではなく、専用のコンパイラを開発して利用者に提供した。「富岳NEXT」では演算装置がCPU (FUJITSU-MONAKA-X)とGPU (NVIDIA製)の2種類になる。GPU部で実行するコードのコンパイルのためには、GPUの機能・性能を最大限に発揮させるために、NVIDIA HPC SDK (ここではNVIDIA CUDA Toolkitを含むものとする)を提供する。CPU部で実行するコードのコンパイルに使用するコンパイラとして、何を提供するかと、それをどのように実現するかを決める必要がある。以降ではそれらについて述べる。

3.3.1.1.2.1.1 候補となるコンパイラ

GPU部で実行するコードのプログラミングでは、GPU部で実行する処理をCPU部の処理から関数・サブルーチンの単位で分離して専用のプログラミング言語で記述する方法(CUDAコード)と、GPU部で実行する処理をCPU部の処理の中にディレクティブやC++テンプレートを用いて埋め込んで記述する方法(GPU offloadingコード)の、2つの方法がある。

NVIDIA HPC SDKに含まれるコンパイラは、CUDAコードとGPU offloadingコードだけでなくCPUコードもコンパイルできる。また、AI/HPCの分野で広く使われているオープンソースのコンパイラとしてLLVMとGCCの系列のものが存在し、CPUコードだけでなくGPU offloadingコードもコンパイルできる。さらに、FUJITSU-MONAKA-Xの機能・性能を最大限に発揮させるために専用のコンパイラを開発・提供することも考えられる。このコンパイラ系列を仮に「富士通コンパイラ」と呼び、その各プログラミング言語用のコンパイラのコマンド名を仮にfjclang (C用)、fjclang++ (C++用)、fjflang (Fortran用)とする。

これらをコンパイル対象コードとプログラミング言語で整理したものを表 3-8に示す。

表 3-8 候補コンパイラ

コンパイル対象コード	コンパイラ系列	コンパイラ(コマンド名)		
		C 用	C++用	Fortran 用
CUDA コード	NVIDIA HPC SDK	(nvcc)[1]	nvcc nvc++	nvfortran
GPU offloading コード (OpenACC/OpenMP/Kokkos)	NVIDIA HPC SDK	nvc	nvc++	nvfortran
	LLVM	(clang)[2]	(clang++)[2]	(flang)[2]
	GCC	(gcc)[2]	(g++)[2]	(gfortran)[2]
CPU コード	NVIDIA HPC SDK	nvc	nvc++	nvfortran
	LLVM	clang	clang++	flang
	GCC	gcc	g++	gfortran
	富士通コンパイラ	(fjclang)[3]	(fjclang++)[3]	(fjflang)[3]

3.3.1.1.2.1.2 CPU部のコンパイラの実現性の要件

ここでは、CPU部のコンパイラの実現性としての要件、すでに存在するNVIDIA HPC SDK、LLVM、GCCが「富岳NEXT」の世代にそれらの要件を満たせるかどうか、そしてどのような場合に富士通コンパイラが必要となるかを示す。ただし、NVIDIA HPC SDKに含まれるnvccコマンドは、内部的に、ソースコード中のGPU部で実行する部分とCPU部で実行する部分を分離し、CPU部で実行する部分はCPU用のコンパイラ(ホストコンパイラ)でコンパイルを行う。ホストコンパイラとしては、nvc++、clang++、g++などを選択できる。そのため、以降では、nvccは扱わず、NVIDIA HPC SDKの代わりに、その一部であるnvc、nvc++、nvfortranの3つをまとめた「NVIDIA HPC compilers」を扱う。

「富岳NEXT」のCPU部のコンパイラの実現性としての要件として、一般的なArmアーキテクチャ用C/C++/Fortranコンパイラの要件以外に、主に以下が挙げられる。

- フロントエンド部において、新しいプログラミング言語規格への継続的な対応
- フロントエンド部において、Arm C Language Extensions (ACLE)での SVE2/SME2 への対応
- バックエンド部において、FUJITSU-MONAKA-X の命令セットアーキテクチャ(ISA)への対応
- バックエンド部において、FUJITSU-MONAKA-X のマイクロアーキテクチャへの対応

3.3.1.1.2.1.2.1 新言語規格対応

「富岳NEXT」は数年以上にわたって運用され、その間にC、C++、Fortran、OpenMPのプログラミング言語規格・プログラミング言語拡張規格が更新され、「富岳NEXT」で実行されるソフトウェアでも新しい言語規格を使用するようになることが想定される。そのため、「富岳NEXT」で提供するコンパイラには、「富岳NEXT」の運用開始後も継続的に新しい言語規格に対応してバージョンアップすることが求められる。NVIDIA HPC compilers、LLVM、GCCは、本書執筆時点で新しい言語規格に対応して継続的にバージョンアップしており、「富岳NEXT」の運用開始後もこの要件を満たせることが想定される。

3.3.1.1.2.1.2.2 ACLE対応

ACLEは、Armアーキテクチャ用のC言語の拡張仕様であり、Armアーキテクチャの発展に伴って様々なデータ型、

[1] nvcc は C++ベースの CUDA C++用のコンパイラであるが、C++と C で互換のある範囲で C ベースとして使用

[2] OSS コンパイラの GPU offloading 機能は OSS サポート状況次第

[3] 富士通コンパイラを提供する場合の仮称

組み込み関数、マクロなどが追加されて続けている。コンパイラやその版数によってACLEの実装状況が異なる。「富岳NEXT」向けには、AI/HPCの分野のソフトウェアに重要なScalable Vector Extension version 2 (SVE2)とScalable Matrix Extensions version 2 (SME2)用のデータ型・組み込み関数に対応したACLEが実装されているのが望ましい。LLVMとGCCでは、本書執筆時点の最新版(LLVM 21.1.8、GCC 15.2)で、それらに対応したACLEが実装されている。NVIDIA HPC compilersでは、本書執筆時点の最新版(NVIDIA HPC SDK 25.11)のACLEの実装において、SVE2には対応しているがSME2には対応していない。NVIDIA HPC compilersのACLEにSME2の対応を入れることによって、NVIDIA HPC compilersでもこの要件は満たされる。

3.3.1.1.2.1.2.3 ISA対応

FUJITSU-MONAKA-XはArmアーキテクチャのISAに準拠している。Armアーキテクチャ ではISAが強化され続けており、各コンパイラでもその対応が進んでいる。コンパイラのISA対応は、そのISAに含まれる命令が使用できるようにすることと、命令を性能最適化などで活用することの、大きく2つに分類される。

LLVMとGCCでは、Arm社が中心となって、命令使用の対応を進めている。命令使用の対応は、ほとんどがアーキテクチャに固有の部分の修正になり、ほかのアーキテクチャに影響を与えることが少ないため、開発コミュニティから拒否されることは少ない。そのため、修正が順調に取り込まれることが多い。NVIDIA HPC compilersは、バックエンドをLLVMベースにしており、本書執筆時点では定期的に新しいバージョンのLLVMにリベースしている。そのため、LLVMで命令使用に対応できれば、NVIDIA HPC compilersでも命令使用に対応できる。

LLVMとGCCでは、LLVM/GCC開発コミュニティ内のArm社を含む部分コミュニティによって、命令活用を進めている。命令活用は、アーキテクチャに固有の部分だけでなく、ほかのアーキテクチャと共通の部分にも修正が必要になることがある。そのため、以下のような特徴を持つ修正は、コンパイラのメンテナンスが困難になりメリットよりデメリットの方が大きいといった理由により、開発コミュニティ内のほかのメンバーから拒否され、取り込まれないことがある。

- 適用ケースが限定的
- 非標準の仕様に依存
- デフォルト動作しない最適化
- 性能向上効果が小さい最適化

NVIDIA HPC compilersも、新しいバージョンのLLVMのバックエンドへのリベースを容易にするため、LLVM開発コミュニティに取り込まれなかった修正を取り込みにくい。したがって、「富岳NEXT」に必須の機能を上記のいずれのコンパイラにも取り込めなかった場合には、それらでは要件を満たせず、富士通コンパイラが必要になる。

3.3.1.1.2.1.2.4 マイクロアーキテクチャ対応

マイクロアーキテクチャ対応もISA対応と同様である。LLVMとGCCでは、既存の仕組みの中でできる修正やほかのCPUに影響を与えることがない修正は取り込まれやすいが、そうでない修正は取り込まれないことがある。
NVIDIA HPC compilersも、LLVM開発コミュニティに取り込まれなかった修正を取り込みにくい。「富岳NEXT」に必須の機能を上記のいずれのコンパイラにも取り込めなかった場合には、それらでは要件を満たせず、富士通コンパイラが必要になる。

3.3.1.1.2.1.3 富士通コンパイラの実現方法

「富岳」では、富士通が開発した専用のコンパイラが提供されたが、運用開始後に新しい言語規格に対応したバージョンアップは行われていない。新しい言語規格に対応したバージョンアップを低コストに行うには、既存のコンパイラをベースにするのが望ましい。2024年度までに行われた次世代計算基盤に係る調査研究(システム調査研究)では、その候補としてLLVMについて調査を行い、LLVMが「富岳NEXT」のベースコンパイラとして適しているとの結論を得た。そのため、「富岳NEXT」で仮に富士通コンパイラが必要な場合、そのベースはLLVMであることが望ましい。

オープンソースソフトウェア(OSS)を活用した一般的なコンパイラのソース・構築方法・配布方法での分類を表3-9に示す。

表 3-9 OSS を活用した一般的なコンパイラのソース・構築方法・配布方法での分類

	分類(便宜上)	ソース・構築・配布	LLVM での具体例
1	公式リリース版	修正なしの OSS ソースを開発元が構築して配布	• LLVM release packages https://github.com/llvm/llvm-project/releases/
2	独自ビルド版	修正なしの OSS ソースを環境に合わせて構築・インストール	• 富岳 OSS 化
3	OS 配布版	OS 開発者が、OSS ソースに最小限の修正(backport など)を加えて、OS 構成に合うように構築し、OS の標準パッケージとして配布	• Red Hat Enterprise Linux 10 (clang/clang++) • Ubuntu 24.04 LTS (clang/clang++/flang-new) • SUSE Linux Enterprise Server 16 (clang/clang++)
4	OSS 投稿準備版	OSS に投稿予定の修正を OSS ソースに先行的に取り込み、それを構築して配布	• Clang for NVIDIA Grace (clang/clang++) https://developer.nvidia.com/grace/clang
5	OSS ベース版	OSS ソースに独自修正を入れ、それを構築して配布	• Arm Toolchain for Linux (armclang/armclang++/armflang) https://developer.arm.com/Tools%20and Software/Arm%20Toolchain%20for%20Linux • AMD ROCm (amdclang/amdclang++/amdflang) https://www.amd.com/ja/products/software/rocm.html • oneAPI (C/C++) (icx/icpx) https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compiler.html • Technical Computing Suite (clang モード) (fcc -Nclang/FCC -Nclang) • NVIDIA CUDA Toolkit (nvcc) https://developer.nvidia.com/cuda/toolkit
6	独自コンパイラ	OSS ソースに独自修正(多くの場合は非公開)を入れたものを部品として活用し、フロントエンドなどは別のものを利用し、独自に構築したものを配布	• NVIDIA HPC SDK (nvc/nvc++/nvfortran) https://developer.nvidia.com/hpc-sdk • oneAPI (Fortran) (ifx) https://www.intel.com/content/www/us/en/developer/tools/oneapi/fortran-compiler.html

これらの分類の特徴を表 3-10に示す。

表 3-10 OSS を活用したコンパイラの各分類の特徴

	分類(便宜上)	特徴
1	公式リリース版	OSS 開発コミュニティがテストしたパッケージを利用できる。配布対象プラットフォームやパッケージ内モジュールは限定されているため、環境に最適とは限らない。 (例: 構築時に-mcpu オプションが指定されておらず、コンパイラ自体の性能、つまり翻訳性能に改善余地がある)
2	独自ビルド版	利用環境に合わせた独自の構築ができる。問題が起きた場合でも、ソースコードは OSS as-is であるため、OSS 開発コミュニティの協力を得やすい。
3	OS 配布版	OS distribution がサポートを行う。OS distribution によっては、パートナーからの feature request を受け付け、将来バージョンに対する OSS 投稿済みの修正が backport される。
4	OSS 投稿準備版	OSS へマージされる見込みの修正だけが取り込まれている。障害修正や一部機能の先行取り込みを行っており、upstream のステージングの位置づけ。
5	OSS ベース版	OSS ソースに対して、目的に沿った独自の(upstream されていない)修正が含まれている。ライブラリは独自のものを含むことがある。独自追加機能、独自チューニングや、独自ライブラリとの連携を行う場合に用いる。OSS 開発コミュニティに拒否されるような機能を取り込むことができる。多くの場合、コマンド名を変えている。
6	独自コンパイラ	独自色の強いコンパイラを構築することができる。ほとんどの場合、コード生成に LLVM のバックエンドを利用しているが、フロントエンドは独自であるため、ユーザービューでは LLVM コンパイラと互換性がない。

「富岳NEXT」で富士通コンパイラが必要な場合は、LLVM開発コミュニティに拒否されたISA対応やマイクロアーキテクチャ対応のための独自修正が「富岳NEXT」で必要な場合である。それ以外はLLVMをそのまま使うことが望ましい。そのため、「富岳NEXT」で富士通コンパイラが必要な場合は、上記の「OSSベース版」とすることが望ましい。また、富士通コンパイラを提供する場合は、LLVMのバージョンアップに伴って、富士通コンパイラもリベースすることが望ましい。

3.3.1.1.2.1.4 富士通コンパイラの要否

富士通コンパイラを提供する場合のデメリットとして以下が挙げられる。

- 「富岳 NEXT」の運用者にとって、独自修正を持ちながら障害修正を行ったり LLVM の新バージョンにリベースし続けたりするには、維持・保守のコストがかかる
- 「富岳 NEXT」の利用者にとって、CPU コード用のコンパイラの選択肢が増え、使い分けが複雑になる。

「富士通コンパイラ」はCPUコードを対象とするため、GPU重視のアプリケーションでの需要は低いと考えられる。基本設計の時点で、性能面でCPUコストが主かつコンパイラ対応が課題となるケースが得られていないため、需要の判断が困難である。そのため、LLVMに対して独自修正を適用した富士通コンパイラは、現時点で必要性を示せない。基本設計では富士通コンパイラは不要とし、詳細設計で要件に応じて再検討を行う。詳細設計では、各WG

の基本設計結果をふまえ、開発項目の詳細検討や性能評価を通じて、課題を具体化して富士通コンパイラの要否を判断するのが望ましい。

3.3.1.1.2.1.5　CPU部のコンパイラのまとめ

以上のことにより、NVIDIA HPC compilers、LLVM、GCCで、CPU部のコンパイラの実現性としての要件を満たせる可能性が高いが、今後の要件次第では満たせない可能性も残る。また、NVIDIA HPC compilersがLLVMをベースにすることから、「富岳NEXT」の世代にNVIDIA HPC compilersがフロントエンド部も含めてLLVMと同等以上の機能・性能を実装していれば、LLVMは不要である。

結論として、「富岳NEXT」のコンパイラとして表 3-8に示すNVIDIA HPC SDK、LLVM、GCCを提供するのが望ましい。ただし、LLVMは、詳細設計においてNVIDIA HPC SDKで機能・性能が十分と判断できれば、不要とする。富士通コンパイラは、基本設計では不要とし、詳細設計で要件に応じて再検討を行う。そこで富士通コンパイラが必要となった場合は、LLVMをベースに「OSSベース版」の形で提供し、LLVMのバージョンアップに伴ってリベースすることが望ましい。

3.3.1.1.3　LLVM 開発

CPUコード用やGPU offloadingコード用のコンパイラとしてLLVM (Clang/Flang)を提供するほか、NVIDIA HPC compilersのバックエンド部分はLLVMのバックエンド部分をベースとして定期的にリベースしており、富士通コンパイラを提供する場合にそれはLLVMをベースとする。そのため、「富岳NEXT」に必要な機能がLLVMに存在しない場合は、LLVMに対しての開発が必要になる。ここでは、詳細設計以降で必要になる、LLVMについて検討・設計・開発すべき項目について述べる。これらの検討・設計・開発は、LLVM開発コミュニティと協力して進めるのが望ましい。

なお、ここに述べる項目のほかに、アプリケーションとの協調設計で要望のあった機能改善や、開発項目の詳細検討や性能評価を通じて見つかったコンパイラの課題についても、詳細設計以降でLLVM開発コミュニティと協力して検討・設計・開発を進める。

3.3.1.1.3.1　フロントエンド

3.3.1.1.3.1.1　言語規格対応

2026年2月にリリース予定であるLLVMバージョン22.1.0で予定されているClang/Flangの言語規格対応の概要を表 3-11に示す。言語規格は、本書執筆時点での最新とその1つ前の2つの版を記載している。

表 3-11　LLVM 22.1.0 の言語規格対応状況の概要

言語規格		対応状況
C	ISO/IEC 9899:2018 (C17)	ほぼすべての機能を実装済み
	ISO/IEC 9899:2024 (C23)	大部分の機能を実装済み
C++	ISO/IEC 14882:2020 (C++20)	ほぼすべての機能を実装済み
	ISO/IEC 14882:2024 (C++23)	大部分の機能を実装済み
Fortran	ISO/IEC 1539-1:2018 (Fortran 2018)	像・共配列に関連した機能以外はほぼ実装済み
	ISO/IEC 1539-1:2023 (Fortran 2023)	開発中

2026年2月にリリース予定であるLLVMバージョン22.1.0で予定されているClang/FlangのOpenMP規格対応の概要を表 3-12に示す。

表 3-12 LLVM 22.1.0 の OpenMP 規格対応状況の概要

OpenMP 規格	C/C++対応状況	Fortran 対応状況
OpenMP 3.1	すべての機能を実装済み	すべての機能を実装済み
OpenMP 4.0	すべての機能を実装済み	大部分の機能を実装済み
OpenMP 4.5	すべての機能を実装済み	開発中
OpenMP 5.0	ほぼすべての機能を実装済み	開発中
OpenMP 5.1	大部分の機能を実装済み	開発中
OpenMP 5.2	大部分の機能を実装済み	開発中
OpenMP 6.0	開発中	開発中

なお、最新のLLVM開発版での言語規格対応状況の詳細は以下のURLに記載されている。

- C: https://clang.llvm.org/c_status.html
- C++: https://clang.llvm.org/cxx_status.html
- Fortran: https://flang.llvm.org/docs/FortranStandardsSupport.html
- OpenMP (C/C++): https://clang.llvm.org/docs/OpenMPSupport.html
- OpenMP (Fortran): https://flang.llvm.org/docs/OpenMPSupport.html

言語規格対応は、LLVM開発コミュニティで活発に進められている。そのため、必要に応じてLLVM開発コミュニティと協力して開発を進める。

また、Flang (Fortran)に関しては、Clang (C/C++)と比較すると、特に比較的最近の言語規格(Fortran 2008以降)への対応の部分で、LLVM開発コミュニティ・LLVMユーザーコミュニティでの品質確認が十分ではない。そのため、Flangの品質確認・品質改善を進める。

3.3.1.1.3.1.2 プログラミングモデル対応

「富岳NEXT」では、マルチコアCPUやGPUを活用するプログラミングモデルとして、OpenACC、OpenMP、Kokkos、standard language parallelism (C++のparallel algorithmsとFortranのDO CONCURRENT構文)、CUDA C++、CUDA Fortranなどを提供する予定である。これらのうち、OpenACC、CUDA C++、CUDA Fortranは主にGPUなどのアクセラレータ向けである。NVIDIA HPC SDKに含まれるコンパイラは、OpenACC、OpenMP、standard language parallelism、CUDA C++、CUDA Fortranをサポートしている。LLVMは、表 3-12に示すようにOpenMPをサポートしており、standard language parallelismはLLVM開発コミュニティで開発が進められている。Kokkosは、NVIDIA HPC SDKに含まれるNVC++とNVCC、LLVMのClangをサポートしている。そのため、コンパイラの観点ではこれらのプログラミングモデル向けの開発項目はない。詳細設計中に開発が必要と判明した場合は、理化学研究所様と協力して開発を行う。

3.3.1.1.3.2 バックエンド

3.3.1.1.3.2.1 FUJITSU-MONAKA-X ISA対応

FUJITSU-MONAKA-XはISAとしてArmv9を採用する。拡張機能を含め、ISAの特徴を活かすようなコンパイラの対応について述べる。採用が決まっている中で性能上重要な拡張としてSVE2とSME2がある。

3.3.1.1.3.2.1.1　拡張機能の対応

SVE2やSME2のように、サポートするかを実装毎に任意に選択できる拡張機能が存在する。コンパイラは、それぞれの拡張機能について、その命令を生成してよいかユーザが指示できなければならない。LLVMはArmの最新の拡張機能まで、そのようなユーザ指示に対応している。また、拡張機能が有効化されたとき、その命令を用いるための最適化の実装が必要な場合もある。例えば、LLVMでは、SVE2で新たに導入された命令を用いて、ヒストグラムを計算するループのベクトル化を可能にするための開発が進められている。アプリケーションベンチマークなどで、FUJITSU-MONAKA-Xで採用する拡張機能の命令を有効に利用できることが分かった場合、その最適化の追加を検討する。

3.3.1.1.3.2.1.2　SVE2の対応

SVE2はベクトル命令セットを規定しており、特徴として実装毎に異なるベクトル幅を選択することができる。通常の命令に比べ何倍ものデータを同時に処理できるため、可能な限りベクトル命令を用いることが望ましい。ベクトル命令を利用するには、主にコンパイラによる自動ベクトル化とベクトル命令に直接対応するイントリンジックを用いて明示的にプログラミングする2つの方法がある。

多くのユーザは明示的なベクトルプログラミングはせず、自動ベクトル化を通してベクトル命令を利用するため、これをサポートすることが望ましい。自動ベクトル化はデータ依存解析やベクトル化の方法を選択するためのコストモデルを含む複雑な機能であり、様々なケースで理想的な命令列を生成できるか検証し、必要なら改善する必要がある。具体的な検証、改善項目は詳細設計以降に検討する。

複雑な構造のループをベクトル化したり、ベクトルレーンを組み替えるなど複雑な最適化をしたりしたい場合は、自動ベクトル化ではなくユーザが明示的にベクトルを考慮したプログラミングをする必要がある。そのための高級言語のインターフェースとしてArm C Language Extensions(ACLE)でSVE2命令に直接対応するイントリンジック関数が規定されている。アセンブリ言語より容易なプログラミングが可能なため、このインターフェースをサポートすることが望ましい。

3.3.1.1.3.2.1.3　SME2の対応

SME2は行列演算を高速に実行するための拡張で、行列型のレジスタと外積演算が主な特徴である。SME2命令の主な利用方法は、イントリンジックやアセンブリなど低レベルなインターフェイスでBLASのような数学ライブラリを実装することであると想定される。アプリケーションはそのライブラリを通してSME2を利用する。SVE2と同様にACLEにSME2のイントリンジック関数が規定されており、これをサポートすることが望ましい。

また、自動ベクトル化のようにコンパイラが自動的に生成する方法、数学的な高レベルの表現(MLIRのlinalgダイアレクトなど)からコンパイラが変換する方法も考えられる。これらの方法は、ユーザの負担は少ないが、複雑な機能であり想定するケースで期待通りの結果が得られるかは注意が必要である。アプリケーションベンチマークなどでこれらの機能が要求された場合、要件に合わせてサポートする範囲や内容を検討する。

3.3.1.1.3.2.2　FUJITSU-MONAKA-Xマイクロアーキテクチャ対応

FUJITSU-MONAKA-Xの性能を最大限に発揮するために、FUJITSU-MONAKA-Xのマイクロアーキテクチャの情報をLLVMに組み込む。また、AI/HPCアプリケーションに現れる典型的なコードパターンに対して効果的な最適化を提供するために、いくつかの最適化パスの強化や新規追加に関する検討を行う。以降、具体的な項目ごとに詳細を述べる。なお、詳細設計期間中に、性能評価などを通じて新たなコンパイラの課題が見つかった場合、適宜対応を検討する。

3.3.1.1.3.2.2.1　-mcpuオプションサポート(-mcpu=fujitsu-monaka-x)

-mcpu オプションで fujitsu-monaka-x を指定可能にし、FUJITSU-MONAKA-X向けの最適化を有効にする手段を提供する。ユーザは必要に応じて、FUJITSU-MONAKA-X向けに最適化されたバイナリを生成したい場合に、このオプションを使用できる。

3.3.1.1.3.2.2.2　コストモデル対応

FUJITSU-MONAKA-Xのマイクロアーキテクチャの情報（レイテンシやパイプラインの情報など）を追加する。これらの情報は、後述する命令スケジューリングやソフトウェアパイプライニングを代表とする、各種最適化パスのパラメータとして使用され、FUJITSU-MONAKA-X向けにチューニングされたバイナリの生成を可能にする。これらの情報は、-mcpu オプションで fujitsu-monaka-x を指定した際に有効になる。

3.3.1.1.3.2.2.3　各種最適化

AI/HPC アプリケーションで FUJITSU-MONAKA-X の性能を最大限発揮するために、LLVM の各種最適化の改善を検討する。例えば、以下の項目が考えられる。

- ループ最適化
- 命令スケジューリング
- pragma 指示子による最適化の制御

なお、実際に開発を行う最適化の選定や、具体的な改善内容に関しては、詳細設計で検討する。

3.3.1.1.3.3　ランタイム

LLVMには独自のOpenMPランタイムライブラリが存在する。それには、CPU-GPU間でのメモリコピー処理など、FUJITSU-MONAKA-Xの準拠するISAやマイクロアーキテクチャ向けにチューニングの余地がある可能性がある。詳細設計で必要に応じてこのチューニングを検討する。

また、NVIDIA HPC SDKにも独自のOpenMPランタイムライブラリが存在する。そのため、必要に応じて、NVIDIAと協力しながら、LLVMでのOpenMPランタイムライブラリのチューニングをNVIDIA HPC SDKにも反映させることを検討する。

3.3.1.1.4　GCC 開発

CPUコード用やGPU offloadingコード用のコンパイラとしてGCCも提供する。ただし、NVIDIA HPC SDKに含まれるコンパイラがLLVMをベースにしていること、富士通コンパイラを提供する場合にそれはLLVMをベースにすること、LLVMはGCCに遜色ない機能・性能を備えていることから、「富岳NEXT」向けの開発としてはLLVMを優先する。ここでは、詳細設計以降で必要になる、GCCについて検討・設計・開発すべき項目について述べる。

3.3.1.1.4.1　バックエンド

3.3.1.1.4.1.1　-mcpuオプションサポート(-mcpu=fujitsu-monaka-x)

LLVMと同様に、GCCでも -mcpu オプションで fujitsu-monaka-x を指定可能にする。

3.3.1.1.5　プログラミングツール

3.3.1.1.5.1　性能プロファイラ

性能プロファイラについては詳細設計で検討する。

3.3.1.1.5.2　デバッガ

デバッガについては詳細設計で検討する。

3.3.1.1.6　そのほかのソフトウェア

Portable Hardware Locality (hwloc)やnumactl/libnumaなど、コンパイラ・プログラミング言語に関連するそのほかのソフトウェア についても、ほかのWGと連携して詳細設計で開発の必要性を検討する。

3.3.1.2　加速部

プログラミング環境ワーキンググループにおいて、以下の事項について報告するとともに、現在開発中の次世代 Flang ベースの Fortran コンパイラについて富岳 NEXT ソフトウェア検討チームにアーリーアクセスを提供した。。

3.3.1.2.1　各種コンパイラ

以下に、NVIDIA の HPC 向けコンポーネントの現行構成および対応状況を示す。バグ修正および新機能追加は顧客要望およびアプリケーション要件に基づき管理され、各コンポーネントの更新はNVIDIAHPC SDK のリリースプロセスを通じて提供される。

3.3.1.2.1.1　C コンパイラ (NVC)

NVC は、NVIDIA GPU ならびに x86 および Arm CPU 向けの C コンパイラである。コマンドライン引数に基づくオプションを用いて、対象プロセッサ向け C コンパイラ、アセンブラおよびリンカを起動する。ISO C11 に準拠し、OpenACC および OpenMP オフロードによる GPU プログラミング、ならびにOpenACC および OpenMP によるマルチコア CPU プログラミングをサポートする。

3.3.1.2.1.2　C++コンパイラ (NVC++)

NVC++は、NVIDIA GPU ならびに x86 および Arm CPU 向けの C++17 準拠コンパイラである。コマンドライン引数に基づき、対象プロセッサ向け C++コンパイラ、アセンブラおよびリンカを起動する。ISO C++17 をサポートし、C++17 並列アルゴリズム、OpenACC および OpenMP による GPU およびマルチコア CPU プログラミングに対応する。C++20 および C++23 への対応は現在進行中である。さらに、C++26 で予定される機能の一部（reflection、mdarray、submdspan、senders）についてプレビューサポートを提供しており、正式規格化後に最終版をサポートする予定である。

3.3.1.2.1.3　Fortran コンパイラ (NVFORTRAN)

NVFORTRAN は、NVIDIA GPU、x86 および Arm CPU 向けの Fortran コンパイラである。ISO Fortran2003 を完全にサポートし、ISO Fortran 2008、2018 および 2023 については部分的に対応する。ISOFortran の並列機能（do concurrent および多数の配列演算組込み関数）、OpenACC、OpenMP によるGPU およびマルチコアプログラミングをサポートするほか、CUDA Fortran による GPU プログラミングにも対応する。

3.3.1.2.1.4　NVCC

NVCC は、NVC++に加え、GCC/G++および Arm 上の ACLE 拡張を含む Clang をサポートする。GCC 向け ACLE 拡張は CUDA 13.2 リリースに含まれる予定である。Clang に対するホストコンパイラサポートは公式

Clang リリースに準拠する。

3.3.1.2.1.5　GCC

GNU Compiler Collection（gcc、g++、gfortran）も C、C++、Fortran をサポートし、追加のコンパイラ選択肢を提供する。CPU ターゲット向け OpenMP ディレクティブおよび GPU ターゲット向け OpenACC および OpenMP オフロードに対応する。

3.3.1.2.1.6　LLVM

NVIDIA のエンジニアリングチームは、LLVM IR の中間層およびその下層における最適化を上流へ統合することに注力し、コンパイラ性能を広範に向上させることを目指している。具体的な取り組み分野としては、依存関係解析およびループ最適化が挙げられる。NVIDIA は、Clang の最新ソースツリーに基づくビルド（top-of-tree build）を提供しており、これにより性能改善を迅速かつ容易に利用可能としている。Clang コンパイラは、CPU および GPU ターゲットに対する OpenMP をサポートしており、OpenACCへの対応も現在進行中である。さらに、NVIDIA は Flang プロジェクトに積極的に貢献している。Flang は、OpenACC、OpenMP、CUDA Fortran および Fortran 言語の並列機能を含む、Fortran 2023 に完全対応したコンパイラとなる予定である。

3.3.1.2.2　富士通 CPU のサポート

NVIDIA HPC コンパイラ（NVC、NVC++、NVFORTRAN）は、FUJITSU-MONAKA-X CPU をサポートする。NVCC は富士通 Clang コンパイラをホストコンパイラとしてサポート予定である。

3.3.1.2.3　並列プログラミングモデル

NVIDIA HPC SDK コンパイラは、ディレクティブ型オフロード、標準言語並列機能、明示的 CUDA プログラミングなど、相互運用可能な複数の並列プログラミングモデルをサポートする。利用者の現状に即した柔軟な相互運用性の提供を基本方針とする。

3.3.1.2.3.1　OpenACC

NVIDIA HPC コンパイラは OpenACC 2.7 を完全サポートし、3.0 以降については顧客需要に基づき部分対応する。GPU オフロードおよびマルチコア CPU プログラミングに対応する。

3.3.1.2.3.2　OpenMP

NVFORTRAN、NVC++および NVC は CPU および GPU 向け OpenMP の一部機能をサポートする。OpenMP 5.0 機能に重点を置き、移植性および拡張性の高いプログラミング環境を提供する。未対応のディレクティブや API コールは可能な限り無視して移植性を確保する。無視することができない場合、コンパイラが一意に解釈できない場合、あるいは不正確な実行となる場合は、コンパイル時、または実行時に適切なエラーを出力する。サポートされている OpenMP 5.0 の target offload 機能の最新リストは、以下の URL を参照。
https://docs.nvidia.com/hpc-sdk/compilers/hpc-compilers-user-guide/index.html#openmp-subset

HPC コンパイラにおける OpenMP 機能のサポートは、顧客からの要望とアプリケーションのニーズによって決定される。そのため、現状サポートしている機能を超えるもの、OpenMP 5.0 以降に追加された機能については、アプリケーションの要件に基づいて、サポートの可否を検討する。

3.3.1.2.3.3　Kokkos

Kokkos のような抽象化プログラミングモデルの利用により、生産性および移植性の向上が可能である。成熟した CUDA バックエンドが提供されており、NVIDIA ソフトウェアスタックとの検証が継続的に行われている。NVIDIA は Kokkos プロジェクトと積極的に協業することで、NVIDIA GPU のサポートを可能にしている。また、OpenACC バックエンドも実験的に存在する。OpenMP および C++スレッドを含む複数の CPU バックエンドをサポートし、CPU と GPU 間の単一ソース互換性を実現する。NVIDIAは CUDA バックエンドに関する複数の改善をアップストリームにフィードバックし、GPU 上での性能向上を図っている。

3.3.1.3　高度化部

3.3.1.3.1　GPU 向けプログラミングモデルの検討

現在、HPCの分野では、以下に示すGPU向けプログラミングモデルが広く用いられている。

- OpenACC
  OpenACC は、異種コンピューティングのための指示文ベースのプログラミングモデルである。2011 年に Cray、CAPS、NVIDIA および PGI によって提案されたが、2026 年現在サポートを継続しているのは実質的に NVIDIA のみである。
- OpenMP
  OpenMP は、共有メモリ向け並列プログラミングのための指示文ベースの API として、1997 年に複数の HPC ベンダから成るコンソーシアムによって提案された。その後、2023 年のバージョン 4.0 において、アクセラレータ向けのオフロード機能が追加された。
- Kokkos
  Kokkos は、CPU、GPU、その他のアクセラレータを対象とする性能移植性のための C++ライブラリである。米国 Exascale Computing Project の中で Sandia 国立研究所によって 2012 年から開発が始められ、2025 年からは Linux Foundation 配下の High Performance Software Foundation の支援を受け開発が行われている。
- Standard Language Parallelism
  ここでは、プログラミング言語の標準規格が備える、ループの並列処理を記述するための構文を、standard language parallelism と呼ぶ。C++17 以降の並列アルゴリズム (stdpar) や Fortran 2008 以降の DO CONCURRENT 構文などが該当する。

性能、書きやすさ、移植性のそれぞれに関し、OpenACC、OpenMP、CUDA、KokkosおよびStandard Language Parallelismを比較した結果を下表に示す。

	性能	**書きやすさ**	移植性	備考
OpenACC	△	○	△	
OpenMP	×	○	○	コンパイラの成熟により、性能は改善すると見込まれる。
CUDA	○	×	×	

Kokkos	△ (バックエンドに依る)	△	○	
Standard Language Parallelism	×	○○	○	

NVIDIA製GPUにおけるサポート状況、性能、将来性を考慮し、富岳NEXTにおいては、OpenACCを中心的なGPU向けプログラミングモデルと定めるとともに、Kokkosも推奨するものとする。なお、それ以外のプログラミングモデルのサポートも行う。

上記に加え、異種システムをプログラミングするオープンな標準であるSYCL [0] 、近年注目を集めているJulia [1] およびそのアクセラレータ向け拡張であるJulia for Accelerators (JACC) [2] についても、今後の動向を注視する。

[0] https://www.khronos.org/sycl/
[1] https://julialang.org/
[2] Pedro Valero-Lara, William F. Godoy, Het Mankad, Keita Teranishi, Jeffrey S. Vetter, Johannes Blaschke, and Michel Schanen. 2025. JACC: Leveraging HPC Meta-Programming and Performance Portability with the Just-in-Time and LLVM-based Julia Language. In Proceedings of the SC '24 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '24). IEEE Press, 1955-1966. https://doi.org/10.1109/SCW63240.2024.00245

3.3.1.3.2 Kokkos

- F2Kokkos

上述の通り、KokkosはC++を対象としているため、Fortranにそのまま適用することはできない。これに対し、KokkosのFortran Language Compatibility Layer (FLCL) [3] は、主に、Fortranの配列をKokkosの多次元ビューに変換する機能を提供する。FLCLを用いることにより、ユーザは、Fortranプログラムに含まれるループを、Kokkosで記述された関数の呼び出しに置き換えることが可能になる。

[3] https://github.com/kokkos/kokkos-fortran-interop

そこで、FLCL、および理研が従来より開発していたソース-to-ソースのコンパイラ基盤であるOmniコンパイラ基盤 [8] に基づき、FortranとKokkosの相互運用のためのツールF2Kokkosを開発する。F2Kokkosの利用例を以下に示す。

1. ユーザは、対象ループ (図 3-13) の直前に parallel_for ディレクティブを挿入する。
2. F2Kokkosは、対象ループを変換するとともに (図 3-14)、Kokkosコード (図 3-15) およびインターフェイス (図 3-16) を生成する。

[8] https://omni-compiler.org/

```
subroutine sub
real :: x(xil:xiu,xjl:xju), y(yil:yiu,yjl:yju), a
!$kks parallel_for data(x,y,a) on(i,j) tile(t1,t2)
do j=jl, ju
  do i=il, iu
    y(i,j) = y(i,j) + a * x(i,j)
  end do
end do
end
```

図 3-13 元の Fortran プログラム

```
subroutine sub
use :: sub_kokkos_mod
real :: x(xil:xiu,xjl:xju), y(yil:yiu,yjl:yju), a
call kokkos_sub0(to_nd_array(x), to_nd_array(y), a,
                 xil, xjl, yil, yjl,
                 il, iu, jl, ju,
                 t1, t2)
end
```

図 3-14 変換後の Fortran プログラム

```
#include <Kokkos_Core.hpp>
#include “flcl-cxx.hpp”
extern “C” {
  void kokkos_sub0(flcl_ndarray_t *nd_array_x,
                   flcl_ndarray_t *nd_array_y,
                   float a,
                   int xil, int xjl, int yil, int yjl,
                   int il, in iu, int jl, int ju,
                   int t1, int t2){
    auto x = flcl::view_from_ndarray<float**>(*nd_array_x);  ! Allocated on CudaUVMspace
    auto y = flcl::view_from_ndarray<float**>(*nd_array_y);
    parallel_for(“sub0”, Kokkos::MDRangePolicy<Kokkos::Rank<2>>({il,jl},{iu+1,ju+1},{t1,t2}),
      KOKKOS_LAMBDA(int i, int j) {
        y(i-yil,j-yjl) = y(i-yil,j-yjl) + (*a) * x(i-xil,j-xjl);
    });
  }
}
```

図 3-15 Kokkos コード

```
module sub_kokkos_mod
use, intrinsic :: iso_c_binding
use :: flcl_mod
interface
  subroutine kokkos_sub0(nd_array_x, nd_array_y, a, xil, xjl, yil, yjl, &
                         il, iu, jl, ju, t1, t2) bind (c)
    use, intrinsic :: iso_c_binding
    use :: flcl_mod
    type(nd_array_t) :: nd_array_x
    type(nd_array_t) :: nd_array_y
    real :: a
    integer(c_int) :: xil, xjl, yil, yjl
    integer(c_int) :: il, iu, jl, ju
    integer(c_int) :: t1, t2
  end subroutine
end interface
end module
```

図 3-16 Fortran インターフェース

今年度では、F2Kokkosの試作として、Kokkosのparallel_forパターンを扱うためのparallel_forディレクティブを開発し、その性能を予備的に評価した。

図 3-17のループを1000回繰り返す処理を、R-CCSクラウドのNVIDIA GH200 Grace Hopper Superchipで実行したときの経過時間を表 3-13に示す。また、比較のため、OpenACCによる等価なコードの経過時間も併せて示す。なお、評価では、NVIDIA HPC SDK 25.9-0およびKokkos 4.7.01を用いた。

```
!$kks parallel_for (x,y,a) on (i,j)
  do j=1, 4096
    do i=1, 4096
      y(i,j) = y(i,j) + a * x(i,j)
    end do
  end do
```

図 3-17 F2Kokkos テストコード

表 3-13 F2Kokkos の評価結果 (sec)

F2Kokkos	F2Kokkos w/ fence	Original OpenACC	nomanagedalloc
2.95	3.38	9.49	3.32

F2Kokkosによる変換を行なった版 (F2Kokkos) の経過時間は2.95秒であったのに対し、比較のために評価したOpenACC版 (Original OpenACC) の経過時間は9.49秒であった。

F2Kokkos版において、変換後のコードに現れるparallel_forパターンの直後にメモリバリア (フェンス) の処理

Kokkos::fence() を挿入したところ（F2Kokkos w/ fence)、経過時間は3.38秒になった。一方で、OpenACC版のコンパイル時に、オプション `-gpu=mem:unified:nomanagedalloc` を指定したところ（nomanagedalloc)、経過時間は3.32秒となった。F2Kokkos w/ fence と nomanagedalloc の性能はほぼ同等であることから、Gh200の統合メモリを活用する場合、F2KokkosはOpenACCに近い性能になる可能性が示された。ただし、parallel_for直後のメモリフェンスの必要性については、さらに調査が必要である。

- 日本語ドキュメントの作成

  Kokkosの日本における普及促進のため、公式ドキュメント [4] の日本語訳を行なった。本作業は、今年度以降にも公式ドキュメントの更新に合わせて継続し、さらに公式ドキュメントにマージすることを目指す。

[4] https://kokkos.org/kokkos-core-wiki/index.html

3.3.1.3.3 OpenACC および OpenMP

ともにディレクティブ型のプログラミングモデルであるOpenACCとOpenMPのGPU向けプログラミング機能は、相互に変換することが可能である。OpenACCからOpenMPへの変換を行う既存のツール [5][6] に加え、OpenMPからOpenACCへの変換を行うツールの検討を行う。

[5] https://github.com/intel/intel-application-migration-tool-for-openacc-to-openmp
[6] J. E. Denny, S. Lee and J. S. Vetter, "CLACC: Translating OpenACC to OpenMP in Clang," 2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC), Dallas, TX, USA, 2018, pp. 18-29, doi: 10.1109/LLVM-HPC.2018.8639349.

さらに、東大で開発された、OpenACCおよびOpenMPへ変換できる統一ディレクティブSolomon [7] のサポートについても検討する。

[7] Y. Miki and T. Hanawa, "Unified Schemes for Directive-Based GPU Offloading," in IEEE Access, vol. 12, pp. 181644-181665, 2024, doi: 10.1109/ACCESS.2024.3509380.

3.3.1.3.4 コンパイラ

各言語のコンパイラとしてLLVMをサポートすることを基本とし、CPU部ベンダおよび加速部ベンダと協力して開発を進める。

一方で、LLVMのFortranフロントエンド (Flang) は現在コミュニティによって活発に開発が進められているが、特に比較的最近の言語規格 (Fortran 2008以降) のサポート品質にはまだ課題が残る。そこで、Fortran 2008言語規格のテストセット (約270本) を開発し、コミュニティ (CPU部ベンダおよび加速部ベンダを含む) へ提供する 。

3.3.1.3.5 プログラミングツール

性能プロファイラ (CPU実行性能、GPU実行性能、通信性能) やデバッガなどのプログラミングツール類については、詳細設計において検討を行う。

3.3.2 数値計算ライブラリ・ミドルウェア

本章では富岳NEXTに具備されるべき数値計算ライブラリ・ミドルウェアについて、基本設計時点で検討すべき内容、詳細設計、開発に向けて必要事項と想定される内容について記載する。

3.3.2.1 数値計算ライブラリ・ミドルウェアの役割

数値計算ライブラリ・ミドルウェアは、科学技術アプリケーションプログラムのみならず各種プログラムに現れる方程式や高度な計算アルゴリズムの中で共通に使われるものとして必要なソフトウェア群として分類される。数値ライブラリ・ミドルウェアは典型的な数値アルゴリズム群をコード化しライブラリの形にまとめた形式として実現される。高度な数学やプログラミング技術が投入され、高性能技術を基盤として実現されている。通常、一般利用者が短期間に同等機能のソフトウェア群を作成することは困難である。利用者からは数値計算ライブラリ・ミドルウェアが提供する高性能かつ高信頼性を根拠に数値実験を行うことになる。したがって、「過去現在そして将来にわたって正確な結果を生成するプログラムコード」に不可欠な数値計算ライブラリ・ミドルウェアの長期間にわたる継続的な利用が期待される。

3.3.2.2 富岳 NEXT で想定されるライブラリ・ミドルウェアの要件

数値計算ライブラリ・ミドルウェアはアプリケーションプログラムの作成には必須であり、上記で述べたようにその役割は利用者が実現困難な数学問題の求解部分である。本質的に計算負荷の大きな中核部分に使われていることから、その性能がアプリケーションに大きく影響する。

数値計算ライブラリの性能指標はさまざま存在するが、実行時間（速度）、消費電力、計算精度などを考慮して設計すべきである。また、「富岳NEXT」システムが想定する大規模な計算では、並列化の程度によって、利用する計算機アーキテクチャや問題サイズへの制限、並列化効率にも影響する。さらに、FortranだけでなくC、C++に加えてPythonやJuliaなどの新言語から呼び出し可能であるべきであり、汎用言語バインディングがサポートされなければならない。さらに、バックエンドのランタイムモデルとの親和性を考慮した設計になっているかなど、最新のプログラミング言語との連携がなされているかの観点も重要である。

「富岳」から「富岳NEXT」への移行を中心に考えた際に、数値計算ライブラリ・ミドルウェアの果たす役割を無視してはいけない。「富岳NEXT」は、前身「富岳」のアーキテクチャからの進化が著しく、CPUのみの構造からCPU+GPUの構成となる。メモリ階層も複雑化することが想定でき、容量と転送速度とのトレードオフ問題を利用者のプログラムだけでなく、数値計算ライブラリ・ミドルウェアのレベルでも考慮する必要がある。さらに、ノード単体の利用だけでなく複数ノードを利用した分散並列の計算機利用を想定しなくてはならない。特に、ノード単体の計算に絞ったとしても、GPU単体または複数GPUを活用した計算構成としなければ高性能を担保しにくい構造となる予定である。CPUとGPUの計算資源を効果的・効率的に利用できるか、現時点のソフトウェアの構造を大きく変えずに、段階的にスムーズな移行が可能か、保守性・継続性の面からも検討し、適切な数値計算ライブラリ・ミドルウェアの設計を進める。

以降、全体システムを「CPU部」とGPUによる「加速部」に分けるとともに全体の統合に必要な「高度化部」を含めた検討事項を個別に分析し、基本設計時点での議論の内容をまとめる。

3.3.2.3 CPU 部

富岳NEXT システムのCPU 部を対象とする数値計算ライブラリの基本設計について記述する。まず、現状分析から課題を抽出し、次に課題に対する対応方針を示す。続いて開発対象ライブラリの選定に関する調査結果を示し、具体的な開発項目を挙げる。

3.3.2.3.1　現状分析

富岳 NEXT を代表的システムとする FUJITSU-MONAKA-X 搭載システムにおいては、システム全体のソフトウェア体系はオープンソースソフトウェア（OSS）を基盤とする方向性であり、数学ライブラリについても OSS を活用することが想定されている。本節では、FUJITSU-MONAKA-X を搭載した次期システム向けの数値計算ライブラリの基本設計を行うにあたって、現状分析をまとめる。本稿においては、A64FX または FUJITSU-MONAKA 搭載システムを従来システム、FUJITSU-MONAKA-X 搭載システムを次期システムとして説明する。

AI 技術の普及において、推論処理では CPU やモバイルデバイスで行うことにより、GPU 依存を減らし、低コストかつ省電力な環境での実行が求められる傾向がみられる。したがって、CPU 上で高速な行列演算を提供することは、次期システム開発における重要な要素である。このような背景を踏まえ、近年の HPC および AI 向けプロセッサでは、行列演算を対象とした専用ハードウェアや命令セットを導入する傾向が顕著である。行列演算の性能がシステム全体の性能を大きく左右するため、各プロセッサベンダーがこの領域の高速化に注力している。このようなトレンドを踏まえると、次期システムの性能を引き出すために、FUJITSU-MONAKA-X で新たに導入される行列演算機構を効果的に活用することが重要となる。

3.3.2.3.1.1　新規 CPU 命令およびデータ型の導入の潮流

近年のプロセッサ開発において、AI やデータ分析ワークロードの性能を向上させるため、特定の演算に特化した新しい命令セットやデータ型を導入する動きが顕著になっている。例として、Arm アーキテクチャにおける SME (Scalable Matrix Extension) や、Intel アーキテクチャにおける AMX (Advanced Matrix Extensions) の導入が挙げられる。これらの命令セットは、AI の学習や推論で中心的な役割を果たす行列積演算を、ハードウェアレベルで支援する。また、bfloat16 や INT8 といった低精度データ型をサポートすることで、倍精度(FP64)や単精度(FP32)演算に比べて、より多くのデータを一度に処理することが可能となる。この背景には、AI、特にディープラーニングの分野で、モデルの学習や推論における演算の多くが、必ずしも倍精度や単精度のような高い精度を必要としないという知見がある。むしろ、データ型を低精度化（例：FP16, bfloat16, INT8）することで、同じシリコン面積でより多くの演算器を実装でき、メモリバンド幅の消費を抑え、結果としてスループットと電力効率を大幅に向上させることが可能になる。このため、AI 分野での要求をうけて、ハードウェアレベルで高精度浮動小数点演算から低精度データ型へのサポートへとシフトする潮流が生まれている。

FUJITSU-MONAKA-X プロセッサでは、SME の後継である SME2 の採用が予定されている。SME2のベースとなる SME では、ベクトルの外積の和で行列積演算を計算する新規命令追加などの機能拡張が行われており、HPC 用途での行列積演算の性能向上や AI の学習や推論で需要が高い bfloat16 といった低精度データ型の行列演算にも対応する設計となっている。

従来の HPC 用途のアプリケーションでは倍精度利用が主流であり、SVE や AVX のようなベクトル SIMD 命令を用いた最適化が各種ライブラリで重点的に行われてきた。AMX と異なり SME は単精度実数を扱えることに加えて、倍精度実数対応についても CPU 実装有無を選択可能である。SGEMM および DGEMM(ただし必要機能が CPU 実装された場合)での SME 活用が性能発揮のための課題となる。

3.3.2.3.1.2　その他の FUJITSU-MONAKA-X 強化ポイント

プロセッサの世代交代においては、命令セットの追加だけでなく、キャッシュ容量、メモリ帯域、コア数といったマイク

ロアーキテクチャの変更も性能に影響を与える重要な要素である。既存のアプリケーションが性能を維持・向上させるためには、新しいアーキテクチャに合わせたチューニングが求められる。FUJITSU-MONAKA-X は A64FX や FUJITSU-MONAKA と比較してキャッシュ構成や容量の変更が見込まれるため、キャッシュブロッキングの最適な設定も変化することが想定される。また、コア数の増加が見込まれるため、多数コアを効率的に利用するための並列化手法に影響を及ぼす。大規模並列時における同期処理のオーバーヘッド考慮や、NUMA アーキテクチャ下でのメモリアクセス最適化が、性能向上につながると考えられる。

3.3.2.3.1.3　数値計算ライブラリ最適化のアプローチ

数値計算ライブラリの最適化においては、対象とする計算分野の特性を理解した上で効果的なアプローチを選択することが重要である。LAPACK の設計思想は、計算量の多い処理を BLAS level-3（行列-行列演算）に集約し、各 CPU アーキテクチャに特化して高度に最適化された BLAS 実装を用いる点にある。したがって、FUJITSU-MONAKA-X 向けの最適化においても、SME 命令の活用やキャッシュ構成を考慮した高性能な BLAS 実装の提供が重要である。

疎行列ソルバでは、非零要素の分布パターンが性能を大きく左右する。最適なデータ格納形式や前処理、計算アルゴリズムは、対象とする問題の物理的背景や離散化手法に強く依存するため、単一のライブラリで多様な疎行列やデータ格納形式に対して最適な性能を提供することは困難である。このため汎用ライブラリの利用は限定的と考えられる。

FFT カーネル計算では要求 B/F 比を抑えるため、プロセッサのレジスタ数に応じて大きな基数（radix）を選択する。しかし、基数を大きくすると最内ループで同時に読み書きされるデータストリーム数が増加し、キャッシュラインの競合による性能低下を招く可能性がある。 そのため FUJITSU-MONAKA-X の特性を考慮したパラメータチューニング等が必要となる可能性がある。

3.3.2.3.2　対応方針

前節の現状分析に基づき、FUJITSU-MONAKA-X 向け数値計算ライブラリ開発における対応方針を本節で示す。FUJITSU-MONAKA-X で導入される新機能を活用し、アプリケーション性能を向上させる。特に、前節で分析した中核課題である行列積演算（GEMM）に対し、SME2 を活用した最適化に重点を置く方針とする。

3.3.2.3.2.1　BLAS ライブラリでの SME2 対応

HPC 用途および AI 用途の双方に対応するため、OSS ライブラリの拡張、あるいは Arm Performance Libraries のようなベンダーライブラリの活用を通じて、SME2 を活用した行列積演算機能を用意する。

3.3.2.3.2.1.1　BLAS ライブラリでの SME2 活用

HPC 用途の既存アプリケーションの継続利用の観点で、FUJITSU-MONAKA-X 向けに最適化された単精度および倍精度実数の行列積演算ルーチンの提供は必須だと考えられる。SGEMM および DGEMM は標準的な BLAS インターフェースで直接的な呼び出しにより利用される他、level-3 BLASルーチン群や LAPACK ルーチンの内部処理としても多用される。

行列積（GEMM）ルーチンについて、SVE2 に加えて SME2 を活用した演算カーネルを開発する。内部で GEMM を用いる主要な BLAS/LAPACK 性能も改善させる方針とする。

3.3.2.3.2.1.2　BLAS ライブラリでの低精度行列積対応

AI 用途の特にディープラーニングでは、計算性能の最大化が重要な課題となっている。現状分析の節で近年の

プロセッサにおける低精度サポートの潮流として述べたように、学習や推論の高速化を実現するために、低精度演算(例：BF16 や INT8)を用いた高速計算が広く採用されている。この背景から、低精度の行列積演算が必要とされている。

これらの演算を効率的に処理するために設計された、Arm SME や Intel AMX といった専用ハードウェア機能を活用することで、低精度行列積演算ルーチンの高性能実装が可能になる。oneMKL ではAMXを活用する関数として cblas_gemm_bf16bf16f32 等を提供しており、当該関数を使用するコードが PyTorch に見られる。

SME を活用するアプリケーションを増やすためには同様のBLAS 拡張を検討する意義があると考えられる。AI 分野で需要の高い低精度データ型に対応した SME2 カーネルを開発し、推論処理等を高速化する方針とする。

3.3.2.3.2.1.3　FUJITSU-MONAKA-X に適した SME2 活用方法の検討

SME2 命令の活用において、単に新規命令を利用すれば十分な高速化が得られるとは限らず、FUJITSU-MONAKA-X における SME2 実装に適した処理方式を検討することが重要となる。

3.3.2.3.2.1.3.1　Streaming SVE モードの特性の考慮

SME 命令は、Streaming SVE モードに切り替えて実行する。Streaming SVE モードでのベクトル長 SVL(Streaming Vector Length) は、通常の SVE モードのベクトル長 NSVL(Non-Streaming Vector Length)と異なる値とすることも SME 仕様としては可能であり、SVL を NSVL 以上にすることが推奨されている。対象とする行列積パラメータに応じて、2 つのモードのどちらの特性が適しているかは異なる可能性があり、処理方式の切り替えなどの検討余地がある。

3.3.2.3.2.1.3.2　マルチベクトル命令の利用

SME では、Streaming SVE モードでのベクトル長 SVL に応じたサイズとなる二次元の行列格納用領域 ZA ストレージが導入され、その中に配置されるデータ型に応じた ZA タイルという処理単位を扱う。ZA タイルは、行列積演算命令のデスティネーションとして機能し、SVE ベクトルレジスタから生成された外積の結果を保持・累積する。メモリとの間でデータをやり取りする場合には、タイルスライスという単位でロード/ストア命令によりアクセスする。ここで、FUJITSU-MONAKA-X は SME2 拡張で導入された複数の SVE ベクトルレジスタをまとめて扱うマルチベクトル命令に対応する見込みであるため、ZA タイルとのデータの移動にはタイルスライス単位でのロード/ストア命令を用いるかマルチベクトル命令を用いて SVE レジスタを経由するなどの複数の選択肢があり、適した方式の検討が必要となる。

3.3.2.3.2.1.3.3　データ型に応じた処理方式の選択

SME の行列積演算で利用可能なデータ型は、表 3-15に示すように CPU 実装の Feature サポートに依存する。表中の non-widening は入力ベクトルと同じデータ型の ZA タイルに足し込む方式であり、widening は入力ベクトルより高い精度のデータ型の ZA タイルに足し込む方式である。データ型に応じて ZA ストレージに含まれる ZA タイルの数が異なることなどにより、カーネルループ内で用いる ZA タイルの数には調整余地がありえる。

表 3-15 SME の行列積演算命令

Feature	入力データ型	出力データ型	命令※
	FP32	FP32	FMOP[AS], non-widening
	FP16	FP32	FMOP[AS], widening, 2way

FEAT_SME		BF16	FP32	BFMOP[AS], widening, 2way
		INT8	INT32	[SU]MOP[AS], widening, 4way
FEAT_SME 前提	FEAT_SME_F64F64	FP64	FP64	FMOP[AS], non-widening
	FEAT_SME_I16I64	INT16	INT64	[SU]MOP[AS], widening, 4way
FEAT_SME2		INT16	INT32	[SU]MOP[AS], widening, 2way
FEAT_SME2 前提	FEAT_SME_F8F16	FP8	FP16	FMOP[AS], widening, 2way
	FEAT_SME_F8F32	FP8	FP32	FMOP[AS], widening, 4way
	FEAT_SME_F16F16	FP16	FP16	FMOP[AS], non-widening
	FEAT_SME_B16B16	BF16	BF16	BFMOP[AS], non-widening

※ [SU]は signed か unsigned、[AS]は Add か Subtract を示す

3.3.2.3.2.2　FUJITSU-MONAKA-X マイクロアークテクチャ向け最適化

SME2 命令の活用に加え、FUJITSU-MONAKA-X のマイクロアーキテクチャ特性に合わせたチューニングを施すことで、BLAS / LAPACK ライブラリの性能を最大化する。既存の FUJITSU-MONAKA プロセッサ向け最適化で得られた知見を基盤としつつ、FUJITSU-MONAKA-X での変更点を考慮する。

FUJITSU-MONAKA-X は A64FX や FUJITSU-MONAKA と比較してキャッシュ構成や容量が変更されると見込まれるため、データ局所性を高めるためのキャッシュブロッキングのパラメータ（ブロックサイズ）を再調整する。各レベルのキャッシュサイズやレイテンシに合わせて最適なブロックサイズを設定することで、メモリアクセスによるボトルネックを軽減し、演算器の稼働率を向上させる。また、FUJITSU-MONAKA-X で見込まれるコア数の増加に対応するため、スレッド並列処理において処理の粒度やスレッド間の負荷分散を調整する。

3.3.2.3.3　開発対象ライブラリの選定

FUJITSU-MONAKA-X 向けの数値計算ライブラリとしては、Arm Performance Libraries のようなベンダ提供ライブラリを利用する方法と、オープンソースソフトウェア（OSS）を基盤として開発する方法が考えられる。前者は開発期間の短縮や利用者サポート面で有利な一方、後者はカスタマイズの自由度やライセンスの柔軟性に利点がある。本開発では OSS を主軸としつつ、性能や開発効率を評価軸としてベンダーライブラリの併用も検討する。以下では、OSS を基盤とするアプローチを取る場合を想定し、主要な BLAS ライブラリである OpenBLAS、BLIS、LIBXSMM について、それぞれの SME対応状況や特徴、普及度を比較評価し、開発の基盤とする OSS の優先度を検討する。

3.3.2.3.3.1　OpenBLAS

OpenBLAS は、GotoBLAS2 の後継として開発されているオープンソースの BLAS/LAPACK 実装であり、多くの CPU アーキテクチャ向けに手動で最適化されたカーネルを提供することで高い性能を実現している。

SME 対応の経緯について、OpenBLAS の開発コミュニティが管理する github リポジトリでの 2025年 9 月時点での issue とプルリクエストを抽出し 表 3-16に示した。Issue 番号#4715 で Apple M4 の SME 対応の要望があり、SGEMM カーネルの対応カーネルコードが提案されているが、Work In Progress という状態で、取り込まれていない。その後、ファイル名に direct が含まれるソースコードが追加されているが、限定的な条件での SME 利用ルートとなっている。また、混合精度 sbgemm を widening 方式の SME 命令を用いて実装が考えられると思われるが、今のところ議論されていない。

それらの開発による、現時点での SIMD 命令活用状況を表 3-17に示す。

表 3-16 OpenBLAS 開発コミュニティでの SME 対応経緯

issue /PR 番号	内容	Opened	Closed /Merged
#4715	[issue] ARM SME 対応の要望（Apple M4 向け）	2024.5.23	未
#4971	[PR] Apple M4 の自動検出で VORTEX として HAVE_SME を有効化	2024.11.12	2024.11.13
#5011	[PR] Armv9-A アーキテクチャ向け SME2 SGEMM カーネル提案（OpenBLAS の構成上の都合でSYMM/TRMM の対応も必要だが、対処できていない）	2024.12.9	未 (WIP 状態)
#5084	[PR] SME1 利用のSGEMM_DIRECT カーネル対応	2025.1.19	2025.2.19
#5222	[PR] DYNAMIC_ARCH の ARMV9SME ターゲットを修正、MacOS 向け SME クエリコードを追加	2025.4.11	2025.5.11
#5324	[issue] test/sblat*が undefined reference によりビルド失敗	2025.6.20	2025.7.9
#5365	[PR] cmake を使用した DYNAMIC_ARCH ビルドで HAVE_SME 設定を修正	2025.7.9	2025.7.9
#5380	[PR] cblas_sgemm 向けSME1 ベースのDIRECT カーネル対応（alpha とbeta 付き）	2025.7.15	2025.7.18
#5414	[issue] test_extensions/test_sgemmt.c が Apple M4 で SME を使用すると fail 判定	2025.8.4	未
#5423	[PR] VORTEX ターゲットからVORTEXM4 を分離、SGEMM_DIRECT のサポート修正	2025.8.18	未
#5429	[issue] Apple M4 で行列積の数値結果が不正	2025.8.27	未
#5450	[PR] cblas_strmm でSME1 利用の strmm_direct カーネルのサポート、結果不正	2025.9.18	未

表 3-17 OpenBLAS での SIMD 命令活用状況

機能	入力・出力データ型	SSE/AVX/AVX2	AVX-512	AMX	NEON	SVE	SME
BGEMM	BF16	△	○	×	△	○	×
SBGEMM	BF16 → FP32	△	○	○	△	○	×
SGEMM	FP32	○	○	–	○	○	△
DGEMM	FP64	○	○	–	○	○	×
CGEMM	複素 FP32	○	△	–	△	△	×
ZGEMM	複素 FP64	○	△	–	△	△	×

3.3.2.3.3.2　BLIS

BLIS（BLAS-like Library Instantiation Software）は、高性能な行列演算ライブラリを生成するためのオープンソースのソフトウェアフレームワークである。特定のアーキテクチャに依存しないソフトウェア階層と、各種 CPU アーキ向けに最適化されたマイクロカーネルを組み合わせることで、様々なプラットフォームで高性能な BLAS ライブラリ生成を目指している。

表 3-18に示すように、現時点では BLIS において AMX や SME への対応は行われていない。

表 3-18 BLIS ライブラリでの SIMD 命令活用状況

機能	入力・出力データ型	SSE/AVX/AVX2	AVX-512	AMX	NEON	SVE	SME
BGEMM	BF16	×	×	×	×	×	×
SBGEMM	BF16 → FP32	×	×	×	×	×	×
SGEMM	FP32	○	○	–	○	○	×
DGEMM	FP64	○	○	–	○	○	×
CGEMM	複素 FP32	○	△	–	△	△	×
ZGEMM	複素 FP64	○	△	–	△	△	×

3.3.2.3.3.3　LIBXSMM

LIBXSMM は、小規模行列積（x86 向け Small Matrix Multiplication）を高速化するための特化型ライブラリとして、Intel 関係者(Hans Pabst 氏, Alexander Heineche 氏)を主要な開発者として開発された。以下のような特徴がある。

- Intel Extension for PyTorch (IPEX)でも利用
- 実装としては Arm や RISC-V 対応も追加されている
- 引数に制約条件あり： A 転置無, α=1, β=0 or 1
- Just-In-Time (JIT)でコード生成を行う
- 種々のデータ型に対応： FP64, FP32, bfloat16, int16, int8

OpenBLAS や BLIS のようにアプリケーションのリンク時にライブラリを変更する利用方法とは異なり、以下のような利用インターフェースとなっている。

- 従来型の関数呼び出しで利用する場合
  - [sd]gemm 呼び出しにプレフィックス「libxsmm_」を追加
  - 対応範囲の引数であれば内部で JIT 動作
  - [sd]gemm 呼び出しのままで intercept する方式は 2025 年 8 月に廃止
  - 単精度・倍精度 GEMM のみに対応
  - sbgemm や bgemm あるいは整数型の関数定義は行われていない
- C++でのテンプレート利用
  - libxsmm_mmfunction<TYPE>でデータ型情報を指定
  - libxsmm_mmfunction クラスのインスタンス生成時にコンストラクタにより JIT コード生成が動作し、GEMM カーネルの命令列がメモリ上に展開される
  - C++インターフェースでは混合精度に対応しない模様
  - C++23 での低精度データ型は考慮されておらず、対応方針は不明（issue #877）
- C 言語から利用
  - libxsmm_create_gemm_shape 関数呼び出しでデータ型情報を指定
  - 入出力のデータ型をそれぞれ指定可能
  - libxsmm_dispatch_gemm 関数呼び出しで JIT コード生成を動作させ、関数ポインタを取得
  - 取得した関数ポインタで関数呼び出しすることにより、行列積計算を実行

SME 対応の経緯について、LIBXSMM の開発コミュニティが管理する github リポジトリでの関連プルリクエストを抽出し、表 3-19に示した。

表 3-19 LIBXSMM 開発コミュニティでの SME 対応経緯

PR 番号	内容	Opened	Closed /Merged
#466	A64FX 対応（参考）	2021.3.29	2021.3.29 (master ブランチ)
#492, #496	M1 向け AMX 対応はマージに至らず	2021.7.24	-
#910	M4 向け、SME 対応(FP32)	2024.10.16	2024.10.25 (feature_m4_sme ブランチ)
#912	#910 関連ファイルの修正	2024.10.28	2024.11.2 (feature_m4_sme ブランチ)
#915	#910 関連ファイルの修正継続	2024.11.5	2024.11.6 (feature_m4_sme ブランチ)

#916	feature_m4_sme ブランチを main ブランチへマージ	2024.11.6	2024.11.8 (main ブランチ)
#945	CPU 検出の修正など	2025.2.20	2025.2.26 (feature_m4 ブランチ)
#954	環境変数 LIBXSMM_AARCH64_USE_NEON 追加	2025.3.24	2025.3.26 (feature_m4 ブランチ)
#949	feature_m4 ブランチをmain ブランチへマージ	2025.3.6	2025.3.26 (main ブランチ)

JIT 生成された FP32 用 SME 演算カーネルを解析してみたところ、ZA タイル 4 つに対して行列積演算命令を連続実行する、OpenBLAS と同様の方式となっていた 。

SME 命令は混合精度や整数 GEMM にも活用可能と思われるが、FP32 のみの対応となっている。表 3-20 に SIMD 命令活用状況を示す。

表 3-20 LIBXSMM ライブラリでの SIMD 命令活用状況

機能	入力・出力データ型	SSE/AVX/AVX2	AVX-512	AMX	NEON	SVE	SME
BGEMM	BF16	△	○	○	×	○	×
SBGEMM	BF16 → FP32	△	○	○	×	○	×
SGEMM	FP32	○	○	–	○	○	○
DGEMM	FP64	○	○	–	○	○	×

3.3.2.3.3.4　開発対象 OSS の優先度検討

Arm SME 命令セットは一部のライブラリで採用が始まっているものの、オープンソースの OpenBLAS では部分的な対応に留まり、BLIS では未対応である。LIBXSMM は SME 対応が進んでいるものの、特定の用途に特化しており汎用的な BLAS/LAPACK 代替とはならない。普及度の面では Linux ディストリビューションへの同梱なども進んでいる OpenBLAS が優勢であり、SME 対応が不十分とはいえ bfloat16 データ型を扱う BLAS 拡張インターフェースも定義されている。開発対象 OSS としては、多様なアーキテクチャへの対応実績から FUJITSU-MONAKA-X 向けの拡張性も見込める OpenBLAS を第一候補とするのが順当だと考えられる。

3.3.2.3.4　開発項目

抽出した課題および対応方針に基づき、FUJITSU-MONAKA-X 数値計算ライブラリ開発における具体的な開発項目を以下に示す。開発リソースと効果のバランスを踏まえ、GEMM 系ルーチンを中心とした BLAS レイヤの最適化を開発スコープとする。なお Arm Performance Libraries 等ベンダ提供ライブラリの併用については、詳細設計で継続検討する。また FFT のチューニングの必要性は、詳細設計で議論する。

3.3.2.3.4.1　オープンソース BLAS ライブラリへの SME2 演算カーネル導入

オープンソース BLAS ライブラリに対して、SME2 演算カーネルの導入を進める。対象の行列積ルーチンは SGEMM および DGEMM(ただし追加の必要機能が CPU 実装された場合)とする。開発にあたっては FUJITSU-MONAKA-X プロセッサの性能特性をマイクロアーキテクチャレベルでも考慮する必要がある。この開発により、主に HPC 用途において行列積計算を行うアプリケーションの性能向上を期待できる。

3.3.2.3.4.2　オープンソース BLAS ライブラリの低精度行列積対応拡充

オープンソース BLAS ライブラリに対して、低精度行列積の対応拡充を進める。対象の行列積ルーチンは、Python 等で用いられる bfloat16 入力/binary32 出力といった BLAS 拡張インターフェースのものとする。開発にあたっては SGEMM 等と同様に、FUJITSU-MONAKA-X プロセッサの性能特性をマイクロアーキテクチャレベルで考慮する必要がある。この開発により、AI 用途の特にディープラーニングにおいて推論アプリケーションの性能向

上を期待できる。

参考文献

[1] “Part 1: Arm Scalable Matrix Extension (SME) Introduction”, Arm Community blogs, May 23, 2024
（ https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-scalable-matrix-extension-introduction）
[2] “Part 2: Arm Scalable Matrix Extension (SME) Instructions”, Arm Community blogs, June 24, 2024
（ https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-sca lable-matrix-extension-introduction-p2）
[3] SME Programmer's Guide Version 1.0
[4] Arm Architecture Reference Manual for A-profile architecture

3.3.2.4 加速部

富岳 NEXT において CPU部(FUJITSU-MONAKA-X CPU) と加速部(NVIDIA GPU) の双方で一貫性のある数値計算環境を提供できるか、既存の富岳向けアプリケーション資産を最小限の修正により GPU 対応および高速化へ適用可能であるかという点について議論を進めた。その内容について以下に示す。

特に重要視されたのは、数学関数・線形代数・疎行列ソルバといった基盤ライブラリの一貫性、信頼性（精度保証・互換性・実装差分）であった。CPU と GPU で ビット単位の完全一致（bitwise identical）までは要求しないものの、指数関数、三角関数、双曲線関数、累乗・Bessel 関数などについては、ULP（Units in the Last Place）保証の範囲内で結果が一致することを基本要件とした。なお、ビット単位の完全一致には、FMA（積和演算）の抑制、最適化の無効化などが必要になり、性能が大きく低下するため現実的ではないという点で、理研および NVIDIA の認識は一致している。NVIDIA HPC SDK が提供する math.h 実装においては、ULP 保証が明示された（デバイス固有の）関数群が存在する[3.2.1]。しかしながら、これらの保証範囲が富士通 CPU 側のコンパイラおよび数学ライブラリ実装における保証範囲と完全に整合しているかどうかは、現時点で FUJITSU-MONAKA-X CPU 上での検証が不可能であるため確認できない。これらの検証は、詳細設計フェーズの適切な時期に実施される予定であり、その結果に応じて実装に追加的な変更が必要となる可能性がある。

参考文献

[3.2.1] CUDA C++ Mathematical Standard Library Function, CUDA Programming Guide v13.1, htttps://docs.nvidia.com/cuda/cuda-programming-guide/05-appendices/mathematical-functions.html#cuda-c-mathematical-standard-library-function

3.3.2.4.1 各種ライブラリ

システム全体で想定される CPU 向けの数値演算ライブラリと、NVIDIA GPU の数値演算ライブラリの対応関係を整理し、以下のようにまとめる。

3.3.2.4.1.1 NVPL — NVIDIA Performance Libraries

NVPL は NVIDIA Performance Libraries の略であり、Arm 64-bit アーキテクチャに最適化されたライブ

ラリセット、HPC アプリケーションを高速化するためのライブラリ群を指す。NVPL はBLAS、LAPACK、FFT など既存の標準数学 APIと互換性を有しており、ソースコードの修正を必要とすることなく既存コードの高速化を可能とする。代表的な構成要素は以下のとおりである。

- NVPL BLAS：標準 BLAS（Basic Linear Algebra Subprograms）の最適化実装
- NVPL LAPACK：密行列演算・固有値計算・線形方程式ソルバ
- NVPL FFT：FFTW API に対応した高速フーリエ変換ライブラリ
- NVPL RAND：高性能乱数生成ライブラリ
- NVPL SPARSE：疎行列向け線形代数ライブラリ
- NVPL ScaLAPACK：分散メモリ環境向け LAPACK 拡張
- NVPL TENSOR：テンソル計算ライブラリ

3.3.2.4.1.2　cuBLAS — CUDA Basic Linear Algebra Subroutines

cuBLAS は GPU 上で動作する BLAS（Basic Linear Algebra Subprograms）ライブラリであり、行列およびベクトル演算を高性能に実行すること可能とする。CUDA Toolkit や NVIDIA HPC SDK の標準構成要素として提供され、行列―行列積、行列―ベクトル積、各種ベクトル演算などをサポートする。

主な特徴は以下のとおりである。

- BLAS Level 1/2/3 API をサポート
- 複数 GPU や混合精度（FP16/FP32/FP64）をサポート
- 低精度・混合精度行列積や AI アルゴリズム向け融合カーネルを提供する拡張 API（cuBLASLt）を提供
  - cuBLASXt：単一プロセスでのマルチ GPU 対応
  - cuBLASLt：柔軟な GEMM API

3.3.2.4.1.3　cuBLASDx — cuBLAS Device Extensions

cuBLASDx は cuBLAS の デバイス側拡張 ライブラリであり、CUDA カーネル内部から直接 BLAS関数（例：行列積 GEMM）を呼び出すことを可能とする。ホスト側 APIであるcuBLAS とは異なり、カーネル内で低レイテンシかつ高性能な線形代数演算を実行可能とする。

主な機能は以下のとおりである。

- カーネル内で一般行列積（GEMM）を実行
- 計算と他の操作（例：メモリアクセス）との融合による性能向上
- Tensor Core および高速ロード命令の活用
- 現在はプレビューまたは早期リリース版として提供

3.3.2.4.1.4　cuSOLVER

cuSOLVER は、cuBLASを基盤とするGPU 向け密行列数値線形代数ソルバライブラリであり、より高水準の線形代数問題に対してLAPACK形式のAPIを提供する。

主な機能は以下のとおりである。

- 密行列線形方程式ソルバ（疎行列機能は cuDSS および AMGX-Next へ移行予定）
- LU 分解、QR 分解、固有値計算、特異値分解
- 単一 GPU およびマルチ GPU 向け構成を提供（cuSOLVERMp）

3.3.2.4.1.5　cuRAND — CUDA Random Number Generation

cuRAND は、GPU 向け高性能乱数生成ライブラリであり、GPU アプリケーションにおける乱数生成を高速化する。

- 一様分布、正規分布、擬似乱数、真乱数など、様々な確率にしたがう乱数をサポート
- CUDA カーネル内での直接生成および利用が可能
- HPC シミュレーション、統計シミュレーション、モンテカルロ法など乱数依存ワークロードに適する

3.3.2.4.1.6　cuSPARSE — CUDA Sparse Matrix Librarycu

cuSPARSE は、GPU 上で疎行列計算を効率的に実行するためのライブラリである。

- CSR などの疎行列格納形式をサポート
- 疎行列–ベクトル積、疎行列–行列積、疎行列操作および形式変換を提供
- 線形方程式解法、グラフ解析、科学技術計算など大規模疎データ処理に利用
- Tensor Core の疎性機能にアクセスするための拡張 API は関連ライブラリ cuSPARSELt にて提供

3.3.2.4.1.7　cuTENSOR — GPU で加速されたテンソル線形代数ライブラリ

n 次元テンソルの縮約、リダクション、要素演算を行うライブラリである。関連する cuTENSORMp ライブラリは、マルチ GPU およびマルチノード環境でのテンソル演算機能を提供する。cuBLAS、cuSOLVER、cuSPARSE、cuFFT、cuRAND は CUDA Toolkit および HPC SDK に含まれるが、これらに関連する GPU 加速ライブラリが別途提供されている。これらは以下の三つのカテゴリに分類される。

1. デバイス拡張ライブラリ：cuBLASDx、cuFFTDx、cuSOLVERDx、cuRANDDx、nvCOMPDx
2. 疎線形方程式系向け線形ソルバ：cuDSS（直接法ソルバ）、AMGX-Next（反復法ソルバ）
3. nvmath-python：単一ノードまたはマルチ GPU・マルチノード環境において、CPU および GPU 上の NVIDIA ライブラリの高度機能へアクセス可能な Python API 群。nvmath-python.device は、Dx ライブラリへのアクセスおよび Python による高生産性カーネル開発を可能とする。

3.3.2.4.2　尾崎スキーム

Ozaki-I スキームは、2025 年 10 月にリリースされた CUDA 13.0 Update 1 以降、cuBLAS ライブラリ内において製品化されている。NVIDIA は、Ozaki スキームの製品化に向けて多大な研究開発投資（例：ESC アルゴリズムおよび ADP フレームワーク）を行っており、これらの取り組みを継続するためのロードマップを有している。

3.3.2.5　高度化部

3.3.2.5.1　開発・整備予定の項目

高度化の観点からCPU、GPU版、CPU+GPUハイブリッド、分散並列環境版への考察を進める。富岳NEXTシステムの設置に必要な数値計算ライブラリの機能を以下にピックアップし、基本設計時点での検討事項と、詳細設計に向けて選定・整備可能なOSSについての洗い出し、さらにはその候補の要件を議論する。より性能面での詳細検討が現時点で必要なものについては次節にて個別議論する。

3.3.2.5.1.1　BLAS:

基本的な数値線形代数のうち密行列についてその機能を提供する基本ソフトウェア群をBLAS（Basic Linear Algebra Subprograms）と呼ぶ。行列とベクトルの基本演算機能を提供する。特に、その機能は理論

演算量の次数によってレベル 1 から 3 に分類される。レベル 1 はベクトル同士の演算やノルム計算（演算量 O(N)）、レベル 2 は行列ベクトル積（演算量 O(N^2)）、レベル 3 は行列行列積機能（演算量 O(N^3)）である。これらの機能は、データ型と機能、対象となるハードウェアアーキテクチャごとに十分に最適化されたソフトウェアとして、各ベンダのプロプライエタリソフトウェアやフリーソフトウェア開発コミュニティから OSS として提供されている。特に、近代的な数値計算ライブラリや数学ソフトウェアでは、数値線形代数レベルの問題の抽象度を保ちつつ、性能最適化されたコードを部品として組み合わせ、高い実行性能と生産性を高めることを行う。富岳NEXT 開発においても、CPU として FUJITSU-MONAKA-X に最適化された BLAS 及び GPU として NVIDIA GPU に最適化された BLAS、さらには複数ノードにまたがる環境下で線形代数的に同等の機能を提供する分散版 BLAS の機能提供が必要である。CPU と GPU 環境それぞれの BLAS 整備については、各担当ベンダが個別に技術動向と機能面での調査を行っており、それを参照とする。一方、ベンダ提供以外の選択肢として、高性能なオープンソースの GPU 向け BLAS である MAGMA ライブラリに含まれる magmablas などが有力である。

また、CPUやGPUのメニーコア化とメニープロセッサ化に伴い、バッチ処理のような多数の軽量タスクの同時並行処理をサポートする必要性がある。BLASのバッチ化は富岳開発において理研がA64FX向けのBatched BLAS generatorを開発済みであり、FUJITSU-MONAKA-X向けには同機能の継続的な提供は容易である。GPUのみならずCPU側でも混合精度演算に対応した演算器のサポートが実現する可能性があり、インターフェースやその実行粒度の制御の検討が重要となる。行列行列積GEMMについては、混合精度計算アルゴリズムとして知られるOzakiスキームやその他の実装技術について議論がある。これらについては、別章を立ててその詳細を記載する。

分散版BLASの実装についてはPBLASやSLATEなどの議論があるが、開発母体との連携なしには富岳NEXTに最適化したライブラリの開発は困難である。その他に、分散版GEMMについてはいくつか知られたアルゴリズムの独自実装が可能であるが、詳細設計時までにはどの選択肢を選択すべきかの議論を収束させる必要がある。C++26にはlinalgパッケージのクラスとしてBLASと同等の機能の正式な取り込みが決定している。ラッパー機能やデータ管理のランタイムとの連携などプログラミング言語環境開発との議論を詳細設計時に行う必要がある。

3.3.2.5.1.2　LIN: 線形ソルバ

連立一次方程式ならびに内部で呼び出される行列の基本分解操作をここではLINとして分類する。密行列の連立一次方程式は演算量がO(N^3)と非常に高価な演算となるため、大規模な演算が行われることはまれであり、疎行列データ構造の反復解法もしくは直接法のアプローチがなされる。しかしながら、1) 逆行列計算を直接扱うことなく実施する方法、2) 基底データの直交化操作のために必要なQR分解、3) QR分解の一種を実現するためのこれスキー分解などは、その利用が明らかに現れることがなくとも必要性の高い機能としてサポートする必要がある。CPU版にはLAPACK、SLATEなどのオープンな実装が見込める。一方GPU版はcusolverとMAGMAに限定されることになる。今後の開発動向を注視しつつ、これらの中から富岳NEXT向けに整備する。

3.3.2.5.1.3　EVD/SVD: 固有値・特異値ソルバ

固有値分解（EVD）と特異値分解（SVD）は、HPC分野だけでなく量子計算機分野での利用も想定されている。京や富岳の時代から、量子物性シミュレーション（いわゆるマテリアルサイエンス領域）において対角化操作を経てエネルギー計算の中核をなしている。近年では、カルマンアンサンブルフィルタによるデータ同化手法の中心的な数値計算として認識されている。固有値計算アルゴリズムは、ヤコビ法やQR反復法など、直交分解操作を中心とした反復法、ストルム列や行列式の慣性、その他行列式に帰着する二分法に分類される。並列計算機向けの並列解法としては、後者の実装による3ステップ方式（1.ハウスホルダー三重対角化、2. Cuppenの分割統治法、3.逆変換）が多くの近代的な数値ライブラリ（例えば、CPU版ではLAPACKやSLATE、分散版ではScaLAPACKやELPA、EigenExa、GPU版ではcusolver）に採用されている。さらに、ハウスホルダー三重対角

化がSYMVを多用するメモリバンド幅制約の欠点を補う2段階アルゴリズム（密帯変換から帯状三重変換を行う手法）がSLATE、ELPA、MAGMAの3つのライブラリで採用されている。実装上の困難さや分散化におけるスケーラビリティ確保が大きな課題として存在するが、開発動向を注視していく必要がある。

3.3.2.5.1.4　FFT: 高速フーリエ変換

FFT（高速フーリエ変換）は、HPC分野の科学シミュレーションの分野において、行列計算に続き2番目に頻繁に使われる数学関数処理とされる。富岳などのCPU向けのFFTライブラリとしては、汎用的なFFTW、HPC向けに高速化されたffte、そして富岳に使われるA64fx向けに高速化されたものとして、ベンダから提供されているSSL2ライブラリなどが多く使われてきた。GPU向けのFFTライブラリとしては、NVIDIAからcuFFT、AMDからrocFFTがベンダによる専用ライブラリが提供されており、高速な計算が可能である。cuFFTについてはCPU向けFFTWからの移行もサポートされている。その他、各所から様々な用途に応じたGPU版FFTライブラリが提供されており、Vulkan / CUDA / HIP / OpenCLに対応することで様々なGPU間でのコード移行が容易で、主にヘッダで構成され、低レベルで高速なVkFFT．非等間隔FFT向けのcuFINUFFT．高速化なTurboFFT．GEMMなどを組み込みAI向けに高速化されたTurboFNO、分散環境向け3D FFTであるWIFFT-GPU．OpenGL Compute Shaderをベースに開発されたGLFFT．研究レベルのものとしてNukadaFFTなどが挙げられる。

富岳NEXTでは、Monaca-X CPUとNVIDIA GPUを含む多数のノードにより構成されているため、富岳NEXTにおいて求められるFFTライブラリの要件として、

1. マルチGPUにて動作すること
2. マルチCPUにて動作すること
3. 可能であればFFTの利用シーンによって、GPU、CPUの振り分けや協動実行ができること

が挙げられる。3.については、新規開発が必要であり大きな課題といえるが、マルチGPU、マルチCPUに対応するものとしては、DOE Exascale Computing Project で開発された分散メモリ＋マルチデバイス対応の高性能FFTライブラリであるheFFTe（Highly Efficient FFT for Exascale）が挙げられる。このライブラリは、HPC向け分散FFTライブラリであり、3D FFTを主対象とし、MPIによるノード間通信込みのFFTを提供している。またバックエンドとして、FFTW、MKL、cuFFT、rocFFT、oneMKLを使用することができ、単一のコードで様々な環境で動作が可能で、ユーザコードの可搬性も用意であると言える。富岳NEXTでは、このheFFTeをベースに開発を進め、ユーザアプリケーションの移行を促すのが良いと提案する。

3.3.2.5.1.5　PRNG: 疑似乱数生成器

軽量かつ高品質、長周期といった性質を兼ね備えるF2線形法、メルセンヌ・ツイスタ法やxorshift系の手法が広く使われるようになっている。並列計算において必要となるのは、ある程度長周期の疑似乱数であることに加え「ジャンプ」という一定回数の状態更新を一度に行う操作である。というのも、単にスレッドごとにシードのみを変えて乱数列を作った場合、スレッド数の2乗で近接のリスクが発生するからである。ジャンプによって十分な距離を確保することでこの危険は回避できる。 F2線形法は特性多項式を用いたジャンプが低コストで可能ではあるが、ルーチンの提供状況は実装ごとにマチマチであり統一的なインターフェースも存在しない。特に2^64を超える幅でジャンプする場合、多倍長での指定が必要となるので統一インターフェースの設計が難しくなっている。

3.3.2.5.1.6　MATH: 数学関数

項目別の必要機能要件として以下のことを挙げる。「第一にサポートすべき関数は何か」、「C99に記載の関数をすべてサポートで良いか？」などを検討する。また、「静的リンク・動的リンクの両方に対応するか？」それにより「数学関数の出力は同じかどうか」なども検討する。現状としてはMSVCは静的リンクか動的リンクかで数学関数の精度

は異なる。さらに、「プログラミング言語によって初等関数の設計は異なるのか、同じなのか」、これは OSS利用ではなく独自開発を行う場合のみの問題である。「速度に関する要求はどれくらいか？レイテンシで測るか、演算量で測る、何かしらの指標・目標値を設定するべきかどうか」も要検討事項である。

精度に関する要求について、精度に関してはULPという指標を示すことが多いが、ULPにも複数の定義があるため、ULPを用いるならば使う定義を決める必要がある。特に良く使用される変域などに特化した高速な関数があることがある。一例を挙げればIntel OneAPI, CUDAである。またFP64よりも高精度なフォーマットでは、全通りの入力に対する精度を検証できないため、その検証方法を定める必要がある。ベクトル化への対応（複数入力）についても何か決めごとはあるか？

現状では、精度についてFP32についてMSVCでは最大 2.92 e+05 ULP, MKLでは 0.93 ULP, CUDAでは1.35e+07であることが知られている。なお精度面の詳細な報告はhttps://members.loria.fr/PZimmermann/papers/accuracy.pdfに記載がある。この情報は半年に 1 度アップデートされる。また、名古屋工業大学の小泉らのグループは、指数関数・三角関数において、FMAを使用して最速・高精度なアルゴリズムを提案している。このような最新のアルゴリズムを導入すべきかを検討する。

CPUとGPUでの相互運用も課題となる。ベンダ提供の高速なライブラリは一般には精度面での互換性はない。しかし移植時やデバッグ時には、（必ずしも完全精度ではなくとも）両者で一致する結果の得られる数学関数が必要となる。幸い、両者の浮動小数点数フォーマットと演算器はIEEE-754 FMAに準拠しており、三角関数や指数・対数関数といった利用頻度の高いものには対しては両者で共通して使える枠組みがあるのが望ましい。そのようなものを用意する場合にも、インライン関数にするのか、LLVM IRでの共通化とするのかなどが検討事項となる。

3.3.2.5.1.7　SPARSE: 疎行列（データ形式を中心に）

3.3.2.5.1.7.1　SpMV

疎行列ベクトル積演算は偏微分方程式の数値シミュレーションの核である連立一次方程式を解く際に現れる重要な基本演算である. Aを行数n, 非零要数nnzの疎行列とするときSpMVはy=αop(A)x+βyのBLAS 2のgemv に似た型の演算である. opは行列の転置をとるか, それに加えて複素共役をとるかを表わす. gemv ではn2の密行列を長さnのベクトルに作用させるが, SpMVでは一行あたりはnnz/n と計算され10～100程度の値となり, gemvよりも演算密度は遥かに低いことがわかる. 疎行列の格納形式は非零要素の行と列の添字と係数の値をそれぞれ配列に格納するCOO形式, 行の情報を圧縮するCSR 形式, 列の情報を圧縮するCSC形式がある. またGPUのメモリアクセスのため, 行を幾つかのグループにわける Sliced Ellpack形式, それを拡張した SELL-C-σがあるが, ブロック化の際に零の値による充填が行われる. また CSR 形式の発展形として各要素がスカラー値ではなく正方の小さい次数のテンソルからなる場合に行と列の場所の記憶量を削減するBSR形式がある. BSR 形式ではテンソルのサイズすなわちブロックサイズと同じ長さのベクトル化が可能であるので, SIMD 演算器を効率よく利用することができる. SpMVは単一の右辺ベクトルに対する演算であるが, 同じ行列成分を複数の右辺ベクトルに作用させる場合は, SpMMとなり行列要素をキャッシュメモリーに保持して再利用することが可能になる. この場合, BとCを密行列としてBLAS 3のgemmに似たC=αop(A)B+βCとなる. また疎行列同士の積演算では演算後に生成される行列は新しい疎行列パターンになるため, 演算の前にパターンを同定し, 結果を保存するメモリを確保しておく必要がある. SpMVは全ての疎行列線形ソルバで用いられること, SpMMは演算密度が高いことから, この二つをライブラリとして提供する. しかしながら, SparseBLAS はライブラリの標準化案がなく, データ形式, 関数名, 引数の構成はベンダによって個別であり, 統一化されていない. Intel社のMKLの三世代分のライブラリ, AMD社のAOCL, nVIDIA社のGPU向けcuSPARSEとCPU向けNVPL SPARSE, ARM 社のARM Performanceライブラリを比較すると行列のデータ形式は幾つかのものがあるが, ライブラリ内部でデータへのアクセスパターンを最適化する inspector-executor モデルが一般的になっている. これらの考察から, 富岳NEXTで実装するライブラリは

次の性能を持つものが望ましい.

- C 言語によるインターフェースで 浮動小数点のデータ型に対応する関数名を個別に準備すること. これは引数のデータ型の指定を単純なものにするためである. (void* 型によるポインタを配列の先頭アドレスに設定し, その型を引数に指定する方法はコンパイル時の型のチェックができないため, ユーザビリティが下る.)
- SpMM が扱う複数のベクトルは 1 次元の配列からなる密行列として記憶し, 整合寸法を引数として渡す. (SpMM は MKL と cuSPARSE および NVPL SPARSE では提供されているが, ARM Performance ライブラリには含まれていない. しかしながら, SME 演算器を利用できるため CPU での最適化実装を図る. C++26 の mdspan によるインターフェースは, 必要があれば C 言語による関数の上部のラッパーインターフェースとして実装する.)
- inspector-executor モデルはライブラリ内部で行列のデータ記憶の構造を変更することで, 最適化をより進めることが可能である. 行列オブジェクトは一般的な CSR 形式で初期化するが, 内部で BSR 形式, あるいは SELL-C-σ に変換することなどが考えられ, 添字の並べ変えを許すとより高度な最適化が可能になる. ユーザが準備した BSR 形式も受け付けるものとする. SpMV/SpMM ルーチンを発行前にユーザが予め並べ替えを実行すると効率が上がるため, 最適化によって生成された並べ替えを保持する整数の配列を取得する機能を追加する.
- 疎行列オブジェクトの生成と棄却, 事前のデータ構造の解析, データの並べ替えのための整数配列の取得, SpMV, SpMM のライブラリのルーチン名は CPU と GPU で共通のものとする. GPU 版では内部のデータ形式は cuSPARSE を継承するが, CPU 版での内部データの形式を基本設計の最終報告までに決定する. これらのルーチンはユーザが直接利用することに加えて, KOKKOS-kernels や PETSc のバックエンドを提供することにも留意する.

3.3.2.5.1.7.2　LIN, EVD, PRECONDのSPARSE機能群について

a. 反復計算による疎行列線形ソルバはPETScがSpMVをベースとするKrylov部分空間法のソフトウェアパッケージを化を行っている. CG/GMRES/BiCGStab などのソルバが提供され, 前処理には代数的多重格子法のGAMG で SOR あるいは Chebyshev多項式によるスムーサを用いるもの, また, 領域分割による部分問題ソルバを用いるadditive Schwarz 法が実装されている.
b. 直接法ソルバにはマルチフロンタル法をFortran90によって実装するMUMPS があり, BLAS 3の利用率は80%強である. PETScの部分問題ソルバとして採用されている.
c. 固有値ソルバは Arnoldi プロセスをRCIによって実装したARPACKがある. SpMVは並列化も含めてRCIによりユーザのルーチンを用いる. SLEPcはPETScの行列演算ルーチンを利用して, Arnoldi 法を実装しておりまた, 経路積分によるアルゴリムも提供している.
d. 非線形ソルバはPETScによるSNESパッケージがあり, trust region Newton法, 非線形CG法, BFGS法などを網羅している.
e. 最適化ライブラリはPETSc のTAO パッケージがあり, 単体法, 主双対内点法などを網羅している. PETScとは独立したパッケージにIPOPTがある.

3.3.2.5.1.8　ミドルウェア：

ミドルウェアは数値計算ライブラリより上位で計算以外のデータ管理やフレームワークを含む非数値計算手続きを現時点では想定している。しかしながら、基本設計時点で明確な定義と選定方針が立てられないのが現状である。また、前節SPARSEでの考察ソフトウェアの一部がミドルウェアとして分類可能との方針もとれるため、記載方法を検

討し今後基本設計の最終報告ならびに詳細設計の時点で内容を明確化する。

3.3.2.6　特に性能最適化を必要とするライブラリ

前節までの議論では、個別の関数や機能に対する方針や決定方法などを進めたが、CPUとGPUもしくはそれらの組み合わせについて開発担当するベンダが整備することや、関連オープンソフトウェアを導入することを前提としている。しかしながら、性能最適化を必要とするものや、国産ソフトウェアとして今後の継続的な開発により国際的なリードが可能なソフトウェアについては、理研主導で富岳NEXT向けの数値ライブラリとして開発を進めたい。基本設計時点では、Ozakiスキームドライバと密固有値ライブラリEigenExaの2点について、開発の過程を詳細化できたものや、詳細設計までに確定すべきものなど、早期開発に必要な内容についてまとめる。

3.3.2.6.1　GEMM: Ozaki スキームドライバ

尾崎スキーム（Ozakiスキーム、またはOzaki-scheme、Ozaki-I/IIとも呼ばれる）は、行列の乗算を低精度演算で模倣する方法である。GPUのAI性能向上にともない、低精度の行列積の性能が大きく向上しており、富岳NEXTで想定されるGPU世代では、倍精度演算と同じ精度の計算も尾崎スキームで実現可能である。これは日本発の新規技術であることをここで明確にしておく。CPUからGPUへ移行する際、失われる可能性がある倍精度演算器の性能補填の一部、あるいはそれ以上の性能向上を目的とし、理研は積極的にOzakiスキームのドライバ開発を推進する。実装については、CPUやGPUの開発企業と連携しながら進めていく。

3.3.2.6.1.1　必要機能要件

Ozakiスキームの実現には低精度行列積和機能が必要であり、それらは際立って高い性能を示す必要がある。特に、現状のプロセッサ開発のトレンドを考慮し、最低限 8ビット整数（INT8）及び 8ビット浮動小数点数（FP8_E4M3）を入力とし32ビット精度程度の出力を行う行列積和アクセラレータを要する。また、これらを用いた行列乗算カーネル呼び出しインターフェースを必要とする。内部演算には倍精度浮動小数点Fused Multiply-Add（FMA）演算も必須である。

3.3.2.6.1.2　現状分析

INT8を用いたOzaki-IはcuBLAS 13.0u2以降にサポートされている。ただし、参照しているアルゴリズムには高速化が提案されており（https://doi.org/10.1177/10943420241313064）、高速化の余地がある。本実装は単精度実数・倍精度実数・単精度複素数・倍精度複素数の行列乗算をサポートしている。高速版アルゴリズムの性能モデル分析により、以下の性能が推定される。

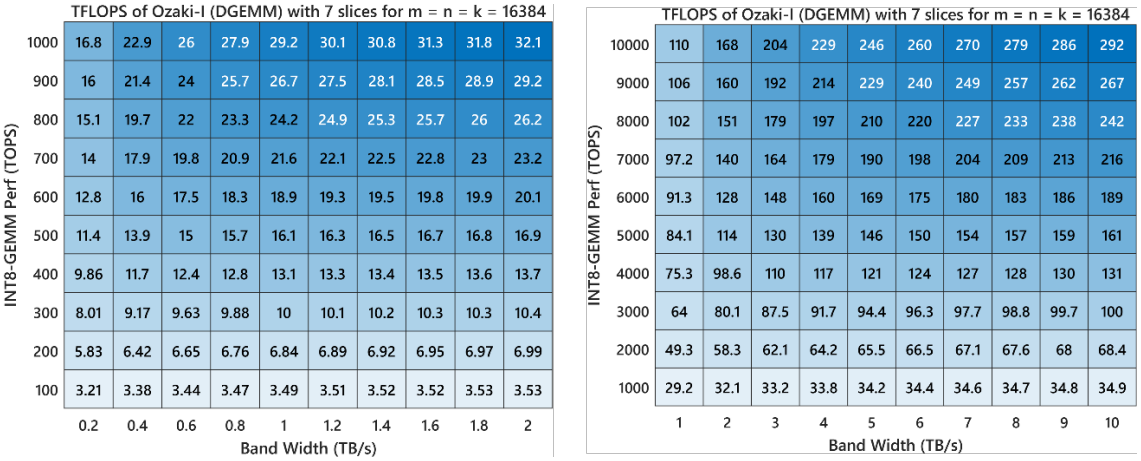


図 3-18 Ozaki-I（DGEMM）の性能予測。縦軸は INT8 行列乗算カーネルの実測性能（TOPS）、横軸は実効メモリバンド幅（TByte/s）。

INT8を用いたOzaki-IIは独自実装のOSSがR-CCS GitHubから公開されている（https://github.com/RIKEN-RCCS/GEMMul8）。本実装は単精度実数・倍精度実数・単精度複素数・倍精度複素数の行列乗算をサポートしている。また、HIP・CUDA両環境をサポートしている。性能モデル分析により以下の性能が推定される。

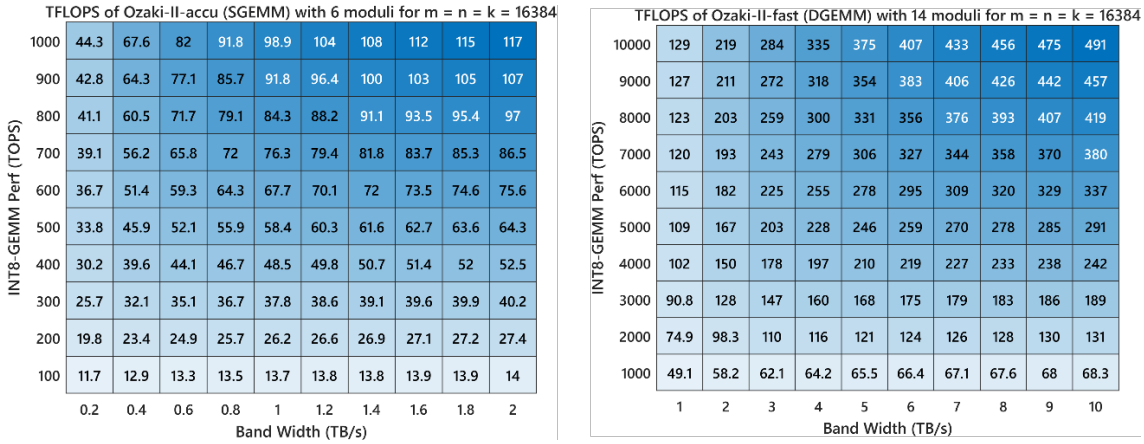


図 3-19 Ozaki-II（DGEMM）の性能予測。縦軸は INT8 行列乗算カーネルの実測性能（TOPS）、横軸は実効メモリバンド幅（TByte/s）。

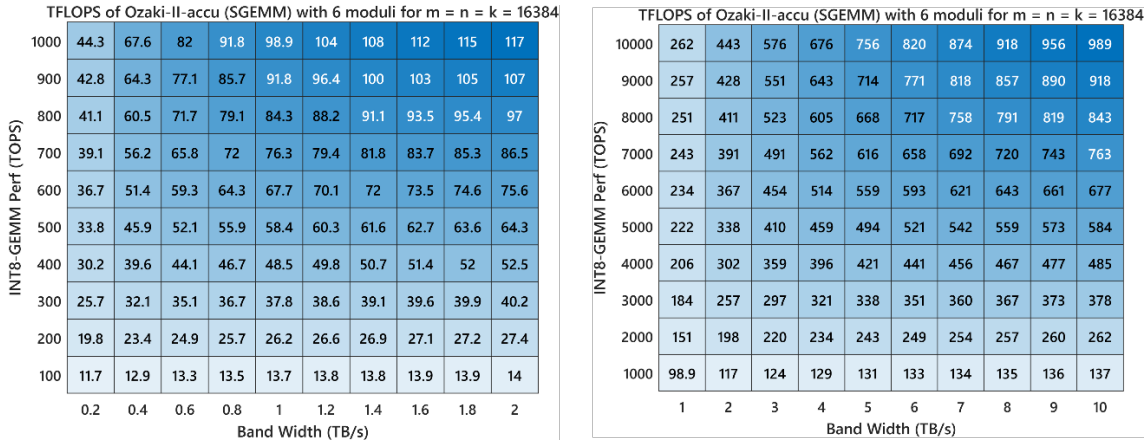


図 3-20 Ozaki-II（SGEMM）の性能予測。縦軸は INT8 行列乗算カーネルの実測性能（TOPS）、横軸は実効メモリバンド幅（TByte/s）。

3.3.2.6.1.3　課題と方向性

NVIDIA Blackwell UltraやNVIDIA RubinのINT8 Tensor Core 性能は前世代から大幅に縮小されており、今後もこの傾向が継続する可能性がある。FugakuNEXTに採用されるGPUも同様の構成となる可能性があるため、8ビット整数（INT8）だけではなく8ビット浮動小数点数（FP8_E4M3）をサポートしたOSSを作成する必要がある。FUJITSU-MONAKA-X CPU 向けのOSSについては、SMEの構成や性能の指標が決定され次第、設計の方向性を定める。

3.3.2.6.1.4　開発項目

FUJITSU-MONAKA-X CPU上に十分な性能で動作するSME2が搭載される場合には、CPUとGPU両者でOzakiスキームをサポートし計算精度と速度面で継ぎ目のない環境整備が可能となる。その実現のためには、FUJITSU-MONAKA-X CPUで動作する8ビット整数・浮動小数点数行列乗算カーネル及びインターフェースが今後必要である。

3.3.2.6.2　EigenExa: 固有値ライブラリ

固有値計算の重要性は、従来のHPCプラットフォームにおける量子物理計算や量子化学計算を中心に、その要求は高い。国内外でも固有値計算のみに限定したオープンソース群が開発されてきた経緯がある。例えば、ヨーロッパを中心としたELPA、CSCCにおけるDLA-future、そして富岳プロジェクトにおけるEigenExaなどが挙げられる。その他、総合的な数値線形ライブラリScaLAPACK, SLATEなども広く利用されている。本節でとり上げるEigenExaは並列計算環境向けの密固有値計算ライブラリとして国産ソフトウェアとして内外で認知されている。富岳プロジェクトの枠を超えて早期のGPU化が望まれていたが、実現には至らなかった。富岳NEXTの開発においてGPU化のみならずGPUクラスタ対応と性能最適化を進める。富岳NEXTに対応するだけでなく、海外の主要スパコン上での動作も考慮し、いわゆる業界標準（De facto Standard）を目指した設計と開発を前提とする。

3.3.2.6.2.1　必要機能要件

富岳NEXT向けの密固有値ライブラリEigenExaに要求される機能は以下のとおりである。

1) CPU、GPU、CPU+GPUの混合環境での任意設定（CPU、GPU、計算ノード分散並列、スレッド並列など）と高性能実行
2) 倍精度と単精度の計算カーネル
3) 荻田・相島の固有値の残差修正反復
4) Ozakiスキームによる混合精度演算制御と性能加速

3.3.2.6.2.2　現状分析

既存のEigenExaは富岳を中心としたCPU単一のクラスタ環境での高並列・高性能実行を達成している。しかしながら、従来のnetlib由来のScaLAPACKやLAPACKなどを内部利用することからFortranコードが含まれている。いくつかのサブプログラムはC、C++化が進んでいるが、現状ではGPUに機能を柔軟にオフロードし、CUDA環境下で高い性能を担保するには、C++への完全移行とともにCUDA化が必須である。さらに、京・富岳は高並列計算機であり、通信アルゴリズムを強く考慮した通信回避型アルゴリズム（Communication Avoiding Algorithm）を採用している。富岳NEXTで想定される、混合精度演算について分析すると、EigenExaは倍精度計算を唯一想定した設計であり、単精度やそれ以外のデータ形式への移行は容易ではない。一方で、富岳実機において倍精度演算相当の計算精度を最終的に担保する形の精度改良の試みは実施されており、SVEによる性能加速のみの状況下でも楽観的な性能改善ができている。

3.3.2.6.2.3　課題と方向性

「現状分析」でも分析した通り、FortranからC++、CUDAへの移行を最優先で実施する必要がある。これは、後述する混合精度演算や複数データ型管理を柔軟に実施することも移行目的に含まれる。次に、富岳NEXTに適したノード間通信を施したマルチノード版への展開を行う必要がある。現状、プログラミング環境や通信ライブラリの基本設計レベルの議論が行われている段階であり、詳細設計時に実装面での課題の解決を図っていくことになる。C++、CUDA移行後のさらなる共通プログラミング言語の移行にも柔軟に対応できるコード管理技術の導入も求

められる。

理研は2003年から2024年にかけて福井大学と連携し、EigenExaの分割統治法群ルーチンのC++化を完了している。C++化のノウハウは他のルーチンや並列計算全般に活用可能である。2025年度の基本設計時には「並列実行制御」、「通信関数」、固有値計算後処理の「ブロックハウスホルダー逆変換ルーチン」のC++化を実施した。C++済みコードの計算結果の整合性と性能劣化がないことも確認済み。詳細設計以降は、前処理のハウスホルダー三重対角化ルーチンのC++化、性能最適化に加えて全体のCUDA移行をAI技術も考慮して進める。

尾崎スキームによる行列積の飛躍的な加速が期待できる状況下で、内部演算が行列積である残差修正反復アルゴリズムとの親和性は高く、従来型のすべての段階で適切に精度良く解くという前提から、適度に粗目の精度でとどめてそれを初期値として、精度改良を進める動的混合精度選択のアプローチを優先的に採用する。FUJITSU-MONAKA-Xにおける混合精度演算機能のサポートは現状未定だが、本アプローチにおいて混合精度を積極的に利用する世界初のCPU+GPU対応版混合精度固有値ソルバとなる。大きな機能変更や実装変更が前提とはなるが、あくまでも内部的な変更であり、CPU、GPUいずれも利用者が大幅な変更を求められるなどのロスは最小化することを最優先に設計と開発を進める。

3.3.2.7　開発ライブラリの性能評価に必要な項目

3.3.2.7.1　ベンチマーク方法の検討

富岳NEXTでは、システムの持続的な性能評価を掲げている。これは今後開発されるHPCシステム環境の設計・評価のみならず、導入されたシステムの維持（障害・修正・バージョンアップ）による問題点の洗い出し、更には、HPCシステム上で動作するプリケーションの開発・性能評価への寄与を目的とした取り組みである。富岳NEXTシステムソフトウェアエリアとしては、言語環境並びに様々なライブラリやツールを開発・共用を目指しており、それらのアプリケーション性能に与える影響を評価することが必要になる。本節では、持続的な性能評価のために必要となるベンチマークへの取り組みについてまとめる。

3.3.2.7.1.1　データセットの準備方法

持続的な性能評価のは、ベンチマークを動かすためのデータセットが必要となる。数値計算ライブラリ・ミドルウェアWGとして提供される成果物の多くは、ライブラリ形式であるため、そのまま性能を評価することは難しい。ユーザは、アプリケーションから数値計算ライブラリを呼び出して使用するため、ユーザアプリケーションからライブラリに呼び出す時に使われる引数が必要になる。またライブラリをシステム性能の評価に用いる場合は、システム性能を引き出せるデータセット並びに、引き出しにくいデータセットなどを用意することが望まれる。また持続的な性能評価を目的とする場合、様々なシステム（CPUやGPU）環境で動作するデータセット用意することが求められる。ベンチマークとして整備した段階でこれらのデータセットを予め用意するのは困難であるため、

1. 1つ典型的なサンプル問題とサンプルプログラム実行環境を作成し、動作に必要なデータセットを作成する。
2. 分散並列有無について典型的なデータセットを作成する。
3. システムのメモリ、キャッシュ容量に応じたデータセットを作成する。
4. いくつかの環境で動作可能なデータセットを作成する。
5. 典型的な問題でアプリケーションを実行した際のライブラリを呼び出す時の引数をダンプし、そのダンプデータからデータセットを作成する。

などの手順が必要となる。1～5までについては、ダミーデータを自動生成できるようにするのが良い。

3.3.2.7.1.2　評価の方法について

　データセット並びに、サンプル問題を用いた実行環境を、次節で述べるベンチマーフレームワーク上に載せて、持続的に様々な環境で性能評価できるようにする。フレームワーク上に載せた評価環境は、定期的に、若しくは何らかのシステム変更・ライブラリの変更をトリガーとして性能評価が可能となっている。ベンチマーク化には、環境変更による誤りを検出するため、チェックサムを出力する必要がある。また性能を評価するため、いくつかの特徴的な区間に分けて時間を計測できるようにすることが望まれる。これらの区間で、ハードウェアカウンタ情報を取得できれば、区間毎の演算・通信・I/O特性などを評価可能になる。フレームワークには、システムパラメータが変った時の性能予測が出来る仕組が組み込まれる予定であり、システムのコデザインに使用可能になる。

　一般に、ベンチマークフレームワークに載せる作業者は、アプリケーション開発者が行うことが多い。しかしライブラリなどは、外部で作成されたものをベースに調整することが多いため、ベンチマークフレームワークに乗せる作業はWGで行う必要がある。

3.3.2.7.2　CB/CI/CD の手法に則ったベンチマーク群の作成

　CI/CDとは、Continuous Integration（継続的インテグレーション）とContinuous Delivery/Deployment（継続的デリバリー/デプロイ）の略で、富岳NEXTでの開発に向けて、システムの持続的な性能評価を念頭にCB（Continuous Benchmarking）が提唱された。アプリケーション開発エリア内ベンチマークWGでは、富岳NEXT向け性能評価プラットフォームとして、BenchKitとBenchParkの2つを用意している。BenchKitはベンチマークWGが独自に用意したものであり、ソースコード並びにデータをGithub上に管理し、構築環境/スクリプト、実行環境/スクリプトを用意することで、様々なシステム環境で性能を取得し、比較が可能になる。導入が容易で、システムに不慣れな一般ユーザでも手持ちのアプリケーションを用いた性能測定が容易であるという特徴がある。BenchParkは調達・性能比較・アーキテクチャ評価を主目的として、DOE（米国エネルギー省）Exascale Computing Project（ECP）が開発したCBプラットフォームであり、HPCアプリ／ミニアプリを「誰でも・同条件で・再現可能」に動かすためのベンチマークフレームワークである。システムごとに環境を構築する必要がある。また構築の可搬性・永続性を担保するため、Spack化が必要になるなど導入ハードルが高いが、システムソフトウェアエリアとして将来の汎用性を考慮するとBenchParkを用いた性能評価・分析を優先して行う。

3.3.3 通信ライブラリ

2030 年頃を見据えた通信ライブラリの基本設計案ついて記述する。標準的な通信ライブラリの OSS を活用し、富岳NEXTのハードウェアアーキテクチャに最適化された通信ライブラリの実現を目指す。

3.3.3.1 全体システム・CPU 部

3.3.3.1.1 通信ライブラリの動向

2023年ごろを見据えた通信ライブラリの周辺動向を調査した結果を示す。

3.3.3.1.1.1 MPI 標準規格

Open MPI とMPICH の比較 MPI ライブラリのOSS として、現在、Open MPI とMPICH が2 大コミュニティとなっている。図 3-21 にOpen MPI [1] とMPICH [2] の通信層の実装比較図を示す。

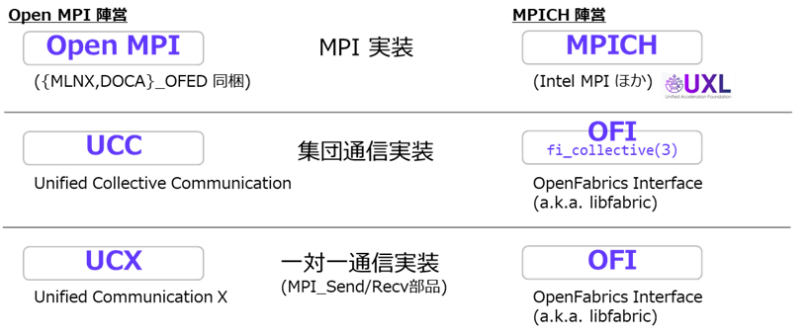


図 3-21 Open MPI と MPICH の通信層の実装比較

MPICH をベースとして使っているサイトがあるが、Open MPI とMPICH を比較 すると、Open MPI は全体に完成度が高く、MPICH は MPI 標準の追随が良好である特徴があるが、 Open MPI には以下の優位性がある。

- スーパーコンピュータ「富岳」までの Fujitsu MPI は Open MPI ベースにしてきており、これまで積み上げてきたソフトウェア資産・ノウハウを富岳 NEXT へ活かすことができる。
- 富岳 NEXT では加速部のベンダが NVIDIA 社に決定していることを考慮すると、現状の NVIDIA 社による Open MPI 陣営への寄与の大きく、特に GPU-aware MPI については Open MPI が先行していることは優位である。
- NVIDIA 社が Open MPI をベースとした製品の提供をおこなっている。

上記から、Open MPI を優先候補とし、詳細設計に向け MPI アプリの評価基盤を早期に固める。

3.3.3.1.1.2 通信ライブラリソフトウェアスタック

図 3-22にシステムライブラリの関係図を示す。MPI は各種基盤ソフトの集積し、システム資源を抽象化して API を提供している。図 3-23にOpen MPI の内部構造を示す。openmpi-5.0 [3] より InfiniBand の通信を担うコンポーネントのopenib (infiniband / RoCE 実装) が削除されたため、Open MPI 自体は、実質、通信のフロントエンドとなった。インターコネクト固有処理は低レベル通信ライブラリ (UCX, OFI) に移行された。この結果、Open MPI において COLL のみが実質的に通信実装を有するが、UCC による置き換えが進んでいる。

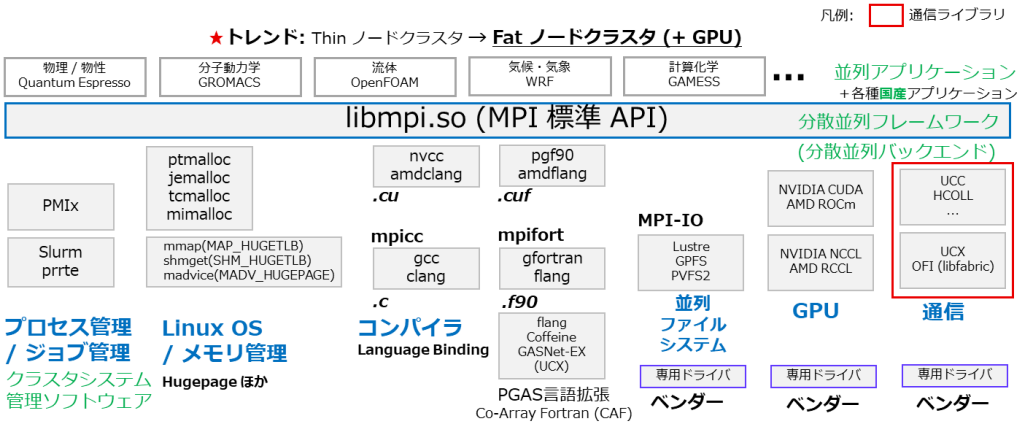


図 3-22 システムライブラリの関係図

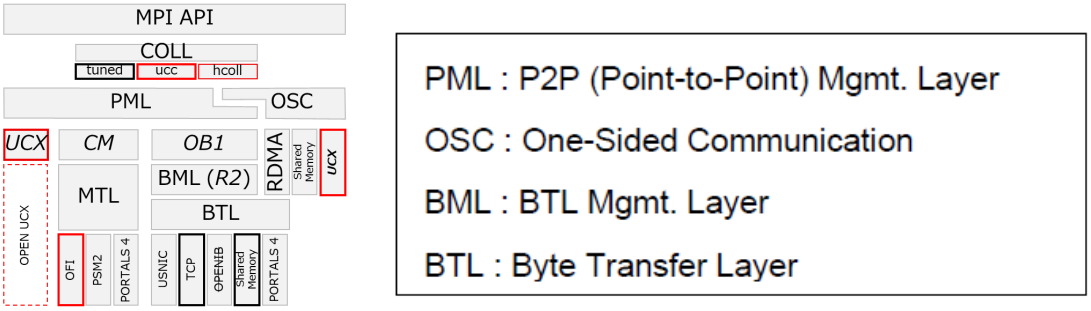


図 3-23 Open MPI の内部構造

3.3.3.1.2　MPI 標準規格動向

Open MPI　のMPI　標準規格 [3] への追従状況を示す。Open MPI はVersion 6.0.0 でMPI 4.1 Standard に対応予定である。Version 5.0 ではMPI 3.1 Standard準拠となっている。

表 3-21 Open MPI の MPI 標準規格への追従状況

MPI Standard	Release Date	Open MPI 対応
MPI 5.0 Standard	2025/06/05	
MPI 4.1 Standard	2023/11/02	openmpi-6.0.0 予定機能
MPI 4.0 Standard	2021/06/09	
MPI 3.1 Standard	2015/06/14	openmpi-5.0.8 (latest)
MPI 3.0 Standard	2012/09/21	

3.3.3.1.2.1　MPI 標準規格動向 - MPI 4.0 Standard

MPI 4.0 Standard (2021/06/09 リリース) [4] では、近年の扱われるデータ量の増大に伴い、int 型の count 引数が十分でなくなり、MPI_Count による「large-count」の API が追加された。これにより、特に集団通信や MPI_File_*() で、サイズ制限が緩和される見込みである。

3.3.3.1.2.2　MPI 標準規格動向 - MPI 5.0 Standard

MPI 5.0 Standard (2025/06/05 リリース) [5] では、異なる MPI 実装の相互動作を可能にするために、標準の ABI (Application Binary Interface) 規格が追加された。この規格により、現状の API (ソース互換) に加え、将来的には ABI (バイナリ互換) が進むことになる。例えば、Open MPI でアプリケーションをビルドしてできたバイナリをMPICH の環境で動作させることができる。

3.3.3.1.3　一対一通信ライブラリ連携

3.3.3.1.3.1　UCX 概要

UCX (Unified Communication X) [6] は、低レベル一対一通信（および一方向通信）ライブラリである。アーキテクチャは図 3-24、表 3-22に示すような構成になっている（https://github.com/openucx/ucx より）。このUCX のコンポーネントのうち、インターコネクト固有部の実装はTransport 層であるUCT(Unified Communication Transport) [7] が担っている。インターコネクト固有部の追加を行う場合にはUCT への個別実装が必要であるが、UCT 側で用意されたAPI を用いて実装することが可能である。

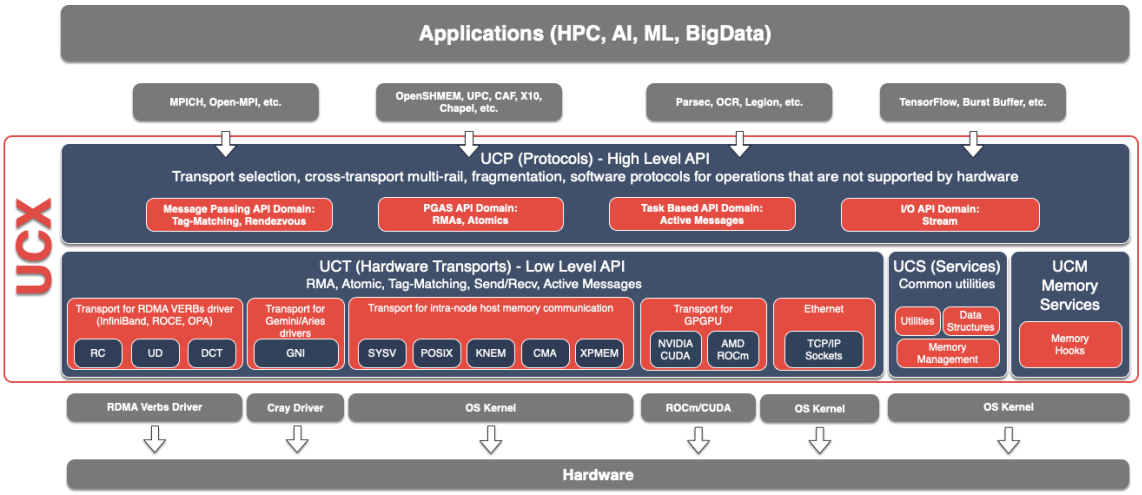


図 3-24 UCX 構成模式図

表 3-22 UCX

Component	Role	Description
UCP	Protocol	Implements high-level abstractions such as tag-matching, streams, connection negotiation and establishment, multi-rail, and handling different memory types
UCT	Transport	Implements high-level abstractions such as tag-matching, streams, connection negotiation and establishment, multi-rail, and handling different memory types
UCS	Services	A collection of data structures, algorithms, and system utilities for common use
UCM	Memory	Intercepts memory allocation and release events, used by the memory registration cache

3.3.3.1.3.2　UCT 概要

UCT (Unified Communication Transport)は、UCX におけるTransport 層を担っており、インターコネクト固有部の実装もUCT 内で行う。UCT のAPI 概要を表 3-23に示す。現状は富岳NEXTのNIC ドライバの仕様が明らかになっていないため、使用すべきUCT のAPI については、富岳NEXT システムのアーキテクチャ確定後に調査・検討が必要である。

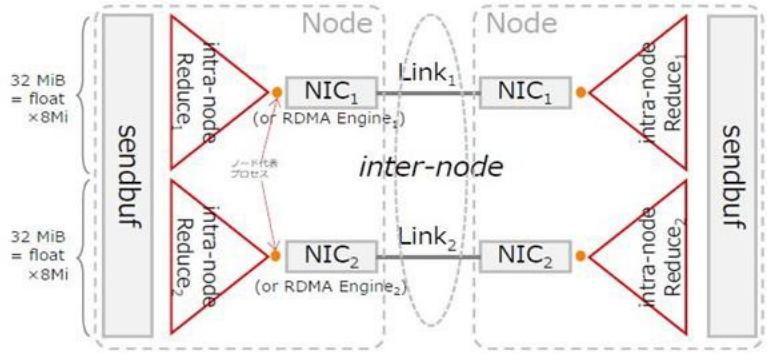
表 3-23 UCT の API 概要

UCT API	**API** 関数の数
UCT Communication Resource	34
UCT Communication Context	7
UCT Memory Domain	14
UCT Active Message	7
UCT Remote memory access operations	6
UCT Atomic operations	6
UCT Tag Matching operations	8
UCT client-server operations	13

3.3.3.1.4　集団通信ライブラリの連携

3.3.3.1.4.1　CPU 部最適化

- 階層型集団通信拡張
  - 集団通信は分散並列アプリケーションでよく使われる通信パターンであり、応用上重要である。MPI ではよく使われる通信パターンを集団通信ルーチン(22 ルーチン)として用意している。集団通信実装は一般に、並列度や転送長などでアルゴリズムを切り替えて、最適なアルゴリズムを選択する。近年、チップレット技術の深化に伴う CPU の大規模化や、NUMA 間ソケット間などのメモリ階層の複雑化が進んでいる。そのため、ノード間通信に比べて高速なノード内通信を前処理として併用する要求が高まっており、ノード内およびノード間で異なるアルゴリズムを使用する「階層型 (Hierarchical) 」集団通信実装が登場している。しかしながら、現状では研究レベルかプロダクション・レベルかは明確ではない。以下に階層型通信の適用例を示す。基本設計では「階層型 (Hierarchical) 」集団通信実装について、基礎的な調査を行った。また、Open MPI および Unified Collective Communication (UCC)における階層型集団通信のフレームワークを示す。Open MPI では階層型集団通信の実装である HAN (Hierarchical AutotuNed) [9] が取り込まれている。集団通信ライブラリの一つである Unified Collective Communication (UCC) [8] おいても、Collective Layer に階層型集団通信の実装 Hier が取り込まれている。



例 1：Multi-Rail 構成での Double Tree による最適化例

	Open MPI	UCC
Name	HAN (Open MPI Hierarchical AutotuNed)	Hier (UCC Hierarchical Collective Layer)
git URL	https://github.com/open-mpi/ompi	https://github.com/openucx/ucc
Path on git	ompi/mca/coll/han/	src/components/cl/hier/

- Open MPI および UCC における階層型集団通信のノード内集団通信のフレームワークを示す。Open MPI には XHC (XPMEM-based Hierarchical Collective) [10] といったノード内専用の集団通信実装も取り込まれており、階層型集団通信実装の部品として動作することが想定されている。

	Open MPI	UCC
Name	XHC (XPMEM-based Hierarchical Collective)	TL/SHM in NVIDIA HPC-X version UCC ?
git URL	https://github.com/open-mpi/ompi	https://github.com/openucx/ucc
Path on git	ompi/mca/coll/xhc/ openmpi-6.0.0 予定機能	Rev. 2.18.0 HPC-X バイナリのみ。ソース非公開

- 階層型集団通信実装本体以外にも、関連するフレームワークとして、ノード内集団通信（縮約演算部）と、

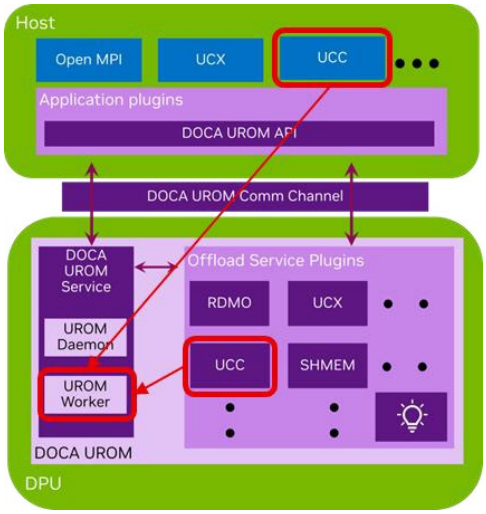
	Open MPI	UCC
Name	Open MPI op/aarch64 MPI_Op 演算コンポ	UCC EC (Execution engine Context)
git URL	https://github.com/open-mpi/ompi	https://github.com/openucx/ucc
Path on git	ompi/mca/op/{base,aarch64}/	src/components/ec/cpu/

ノード内集団通信（プロセス間通信部）があり、最適化を行える可能性がある。

	Open MPI	UCC
Name	SMSC (Shared Memory Single Copy)	? (製品版のみで、ソース非公開のため不明)
git URL	https://github.com/open-mpi/ompi	?
Path on git	opal/mca/smsc/xpmem/ btl/sm 等と共通部	?

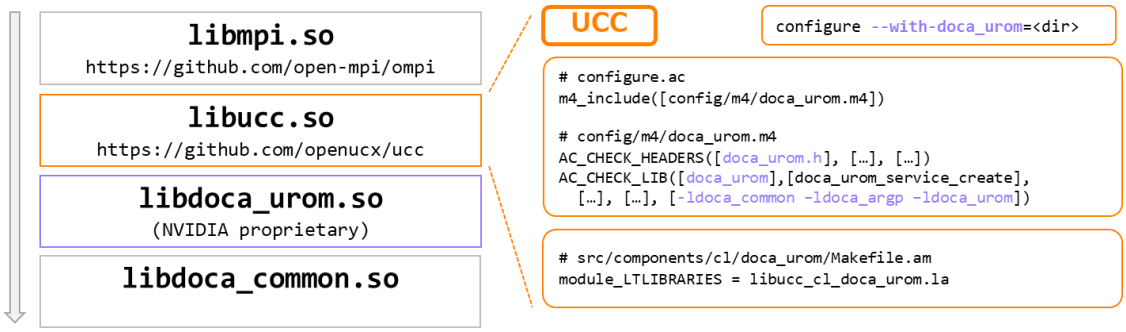
- ネットワーク・トポロジ最適化
  - スパコンセンターのインターコネクトやネットワーク・トポロジは、システムに固有となる場合が ある。スーパーコンピュータ「京」やスーパーコンピュータ「富岳」では、Tofu インターコネクトや Tofu インターコネクト D のような固有の NIC およびトーラス・メッシュのネットワーク・トポロジを持ち、複数の NIC と複数の経路を同時に使用した Trinary-X のようなインターコネクトに最適化されたノード間の集団通信アルゴリズムをサポートしていた。現状ではインターコネクトやネットワーク・ トポロジは、関連 WG で検討中であり、採用ハードウェアや技術仕様は未決定である。このため、イ ンターコネクトに最適なアルゴリズム検討は、各種仕様決定後の検討となる。基本設計では、集団通信のオフローディングや、システムまたは通信スイッチのポートに設置された網内計算 (in-network computing) デバイスの可能性として、UCC の

DOCA UROM （Unified Resource and Offload Manager）による DPU オフロードを一例として基礎的な調査を行った。NVIDIA DOCA UROM [11] は、HPC や AI の並列計算タスクの一部を、ホスト CPU から DPU にオ フロードすることができる。更に、UCC では DPU への「集団通信」一括オフロードの仕組みが取り 込まれており、集団通信処理を DPU へオフロードすることができる。



左図は、NVIDIA DOCA UROM の DPU オフロード模式図に NVIDIA DOCA UROM の DPU オフロード模式図を示す。図中の赤枠と赤矢印は、UCC でホストの「並列計算タスクの一部」を DPU へオフロードする ことと、「集団通信処理」をオフロードすることを表している。

下図に UCC と NVIDIA DOCA UROM の呼び出し階層構造を示す。DOCA (ソフトウェア) 自体は NVIDIA 独自のもので内部は明らかにされていない。



- デバイス固有最適化
  - 近年の普及品の NIC においても RDMA（Remote Direct Memory Access）やパケット振分けを細分化するタグマッチング等の技術が利用可能である。動作条件を指定可能なトリガーつき要求操作や in-network 機能との連携等、様々な固有の拡張を有する場合も多いが、物量が少なく、十分に揃えられない場合は、機能が使用できないことも考えられる。NIC 等の通信デバイスでは一般には、当該デバイスベンダーが UCX や OFI といった低レベルの一対一通信ライブラリでの基本的なアクセスを 提供するが、固有拡張については集団通信レベルでの専用実装が必要になる場合が多い。また、物量の制約から特定の条件下でのみの集団通信サポートとなる可能性もある。 現状では関連 WG で検討中であり、採用ハードウェアや技術仕様は未決定である。このため、採用ハードウェアやネットワーク・トポロジへの最適化検討は、各種仕様決定後に検討する。仮に普及品 と異なる部材に決定し、物理層より上位層に影響が及ぶ場合、通信ドライバや低レベル一対一通信ライブラリの一部（典型的には、インターコネクト固有部）に影響が出る可能性が高い。NIC 等の通信デバイス固有の機能を用いた最適化は、機能の範囲・効果・物量を見定めつつ、集団通信の最適化実装の可否を通信ライブラリ WG で判断する必要がある。

3.3.3.1.5　他ソフトコンポーネントとの連携

- システム固有拡張の統合

- 富岳 NEXT 時代の通信ライブラリは OSS を活用し、必要な機能があれば OSS コミュニティへフィードバックしていくことが基本的な開発方針であると考えるが、スパコンセンターのインターコネクトやネットワーク・トポロジに対する最適化実装は、システムに固有の拡張となる可能性があり、独自の集団通信実装にあたっては、別管理のソース等を維持し続ける必要がある可能性があると考える。Open MPI では依存関係の少ない Modular Component Architecture を採用しており、例えば、各種集団通信は mca/coll インターフェースとして実装されている。さらに、集団通信ライブラリの一つで ある Unified Collective Communication (UCC) ではプラグイン機構を提供している [12]。UCC では、Transport Layer Collective Plugins (TLCP) と呼ばれるプラグイン機構を通して、外付けの集団通信実装バイナリを動的にロード可能である。環境変数でプラグインの優先度を調整することも可能である。基本設計では別管理のソース維持の対応として、TLCP を一例としてあげ、基礎的な調査を行った。

```
$ cd ucc-1.5.0
$ ./autogen.sh
$ ./configure --with-tlcp=ucp_example ...
$ make
$ make install
```

UCC プラグイン (TLCP) の構築
configure --with-tlcp で “,” で連結してプラグインを明示的に指定するか、特殊な“all” か、いずれかで有効化

```
$ mpirun -n 2 ¥
  -mca coll_ucc_enable 1 -mca coll_ucc_priority 100 ¥
  -x OMPI_UCC_TLCP_UCP_EXAMPLE_TUNE="alltoall:0-inf:inf" ¥
  osu_alltoall
```

UCC プラグイン (TLCP) の利用
環境変数でプラグインの priority を調整して有効化。但し環境変数名 *foo*_TUNE はプラグイン実装依存

- システムメモリとの連携
  - Linux OS は仮想メモリアドレスから物理メモリアドレスへの変換のために主記憶上に存在するアドレス変換テーブルを管理する。さらにアクセスを高速にするため、CPU のもつ Translation Look-aside Buffer (TLB) を活用する。しかし TLB のエントリ数は有限であり、探索に失敗した場合、 TLB ミスが発生する。この場合、ハードウェアがメインメモリ上のアドレス変換テーブルを探索する 処理（ハードウェアページテーブルウォーク）が必要となり、TLB にヒットした場合に比べて大きな コスト(時間)がかかる。このためワークロードが消費する TLB のエントリ数の削減のため、1 つの TLB エントリで表現できるメモリ量を数百倍～数千倍という単位で増加させるラージページというハードウェア機能がある。スーパーコンピュータ「京」やスーパーコンピュータ「富岳」では、それをアプリケーションから容易に使用するため libmpg ライブラリと呼ばれる専用のラージページライブラリが提供されてきた。OSS 化への切り替えに伴い、libmpg ライブラリの代替となる手段を検討する必 要がある。 基本設計では、既存の OSS によりラージページを利用可能にする手段の一例として、glibc の malloc 関数 [14] で明示的に Hugepage（Linux 上でのラージページの実装 [13]）を利用するように制御する方法を示す。MPI から Hugepage を有効化するには、環境変数 GLIBC_TUNABLES で glibc.malloc.hugetlb を有効にして MPI プログラムを起動することで Hugepage を利用できる。以下に MPI プログラム実行時の Hugepage の有効化指定例、glibc.malloc.hugetlb に指定する値についての説明を示す。Hugepage を運用で利用するか否かについ て、詳細設計において関連 WG で検討する。

mpirun での指定方法

```
$ mpirun ¥
        -n 4 ¥
        --report-bindings ¥
        --map-by ppr:4:node:pe=72 ¥
        -x GLIBC_TUNABLES="glibc.malloc.hugetlb=2" ¥
        ./mpi-hello.out
```

Tunable: glibc.malloc.hugetlb

This tunable controls the usage of Huge Pages on malloc calls. The default value is 0, which disables any additional support on malloc.

Setting its value to 1 enables the use of madvise with MADV_HUGEPAGE after memory allocation with mmap. It is enabled only if the system supports Transparent Huge Page (currently only on Linux).

Setting its value to 2 enables the use of Huge Page directly with mmap with the use of MAP_HUGETLB flag. The huge page size to use will be the default one provided by the system.

A value larger than 2 specifies huge page size, which will be matched against the system supported ones. If provided value is invalid, MAP_HUGETLB will not be used.

チューナブル・パラメータ: glibc.malloc.hugetlb

このチューナブル・パラメータは、malloc 呼び出しにおける Huge Page の使用を制御します。デフォルト値は 0 で、malloc の追加サポートは無効になります。

値を 1 に設定すると、mmap によるメモリ割り当て後に madvise と MADV_HUGEPAGE の使用が有効になります。これは、システムが Transparent Huge Page をサポートしている場合にのみ有効になります (現在は Linux のみ)。

値を 2 に設定すると、MAP_HUGETLB フラグを使用して mmap で Huge Page を直接使用できるようになります。使用される Huge Page サイズは、システムによって提供されるデフォルトのサイズになります。

2 より大きい値は Huge Page サイズを指定し、システムがサポートするサイズと比較されます。指定された値が無効な場合、MAP_HUGETLB は使用されません。

https://sourceware.org/glibc/manual/2.42/html_mono/libc.html

以下に、Hugepage の運用上の既知課題を示す。Hugepage には、ラージページ専用のメモリプールを使用した Hugepage プール型(事前予約型)と、アプリケーションからの割当要求があった時点で OS の管理するフリー(未使用)メモリからラージページを切り出し、解放要求時に OS フリーメモリに戻す Hugepage 非プール型(一時利用型)が存在する。Hugepage プール型は未使用の(空き)ラージページの探索空間は一般にせまく、割当・解放時のコストはほぼ一定で低コストである一方、アプ リケーションがラージページを使用しなくても事前割当分はノーマルページとして利用することはできないため、メモリ利用効率は低下しやすい。他方、Hugepage 非プール型はノーマルページのフリーメモリと共用利用するため、メモリ利用効率は向上する。しかしながら、ラージページ内は物理連 続領域を獲得する必要があり、断片化の状況により急激に探索コストが増大することも知られている。 更に富岳 NEXT では、GPU Unified Memory などメモリの使い方の変化により、新たな課題が出現する可能性がある。詳細設計で課題を抽出した場合は、関連 WG へ検討依頼をおこなう。

	Pros.	Cons.
HugePage プール型 (例: nr_hugepages)	malloc(3) 等の呼出し場合、毎回 OS の free メモリから、例えば、連続 2 MiB の物理連続領域(ページ) × N 個の切り出しのコストがぶれない (プール内の単純探索で済む)	プール分のメモリは、OS の free メモリには戻らないため、アプリに依っては Normal Page が不足
HugePage 非プール型 (例: nr_overcommit_hugepages)	十分にメモリがある場合、Huge Page と Normal Page がアプリの必要量に合わせて柔軟にフィット	malloc(3) 等の呼出しの際、メモリ状況に寄り探索コストが増減。性能ブレが顕在化する恐れがある ※1

※1 OSの free メモリが不足気味、かつ断片化が進んだ状況では、頁分の連続物理領域が獲得できず、OS が極端なスローダウンを招く恐れもある

3.3.3.1.6 言語連携

3.3.3.1.6.1 Fortran 言語バインディング

MPI 規格においてFortran 言語バインディングはMPI-1.0 から定義されており、Open MPI を含む各種MPI 実装でサポートされている。Fortran 言語バインディングの利用方法は”USE mpi_f08”、”USE mpi”、”INCLUDE mpif.h”の3 種類があるが、Fortran 規格に準拠しているのは”USE mpi_f08”のみであり、これを利用することが強く推奨されている。それぞれの利用方法の特徴は以下のとおりである。

- “USE mpi_f08” は、Fortran 2008 以降の規格をサポートしており、かつ、Fortran 規格に準拠している唯一の利用方法となっている。この方法が強く推奨されている。

- “USE mpi”は、Fortran 2008 より前の規格をサポートする位置づけの利用方法だが、Fortran 規格とは整合しておらず、後方互換のために存在している。
- “INCLUDE mpif.h”は MPI-1 規格の時から存在し、Fortran 77 規格をサポートする位置づけの利用方法である。しかし、MPI-3.0 で非推奨となり、MPI-4.1 規格では廃止予定の位置づけに変わった。

3.3.3.1.6.2　C 言語バインディング

MPI 規格において、C 言語バインディングはFortran 言語バインディングと同様にMPI-1.0 から定義されており、Open MPI を含む各種MPI 実装でサポートされている。また、Fortran とは異なって言語の基本的な仕様が安定しており、利用方法に関する特別な注意事項もない。

3.3.3.1.6.3　C++言語バインディング

MPI規格におけるC++言語バインディングは、1997 年にMPI-2.0 で導入された。しかし、2009 年にMPI-2.2 で非推奨化され、2012 年にMPI-3.0 で完全にMPI 規格から削除された。規格から削除されるに至った主な要因は以下のとおりである。

- C++言語バインディングは、C 言語バインディングに対して付加価値が少ない。
- メンテナンスコストが高い。
- C++の名前修飾が標準化されていないために、バイナリ配布が困難である。

また、Open MPI では2023 年にリリースされたv5.0.0 でC++言語バインディングがサポート対象外となった。Open MPI v5.0.0 以降でconfigure オプション “--enable-mpi-cxx” を指定した場合、 configure エラーとなってビルドに失敗する。なお、Open MPI v4.1.8（2025 年12 月時点でサポート中の前世代リリース）ではC++言語バインディングをサポートしている。したがって、少なくともOpen MPI v4.1.x がサポートされているうちは、2つの世代を使い分けることにより、C++言語バインディン グに対応すること自体は可能である。以上のとおり、C++言語バインディングはMPI 規格とOpen MPI 最新版の両方で削除されている。 このため、富岳NEXT ではC++言語バインディングをサポート対象外とするように見直す必要があると考えられる。

3.3.3.1.6.4　Java 言語バインディング

Java 言語バインディングはMPI 規格には存在せず、Open MPI 固有の仕様となっている [17]。Open MPI v5.0.9（2025 年12 月時点での最新版）でもconfigure オプション “--enable-mpi-java” が存在しており、サポート対象となっている。 Java に関しては、ライセンス関連でOracle JDK（製品）が紛れ込まないように最大のリスク管理をする必要がある。

- 課金体系 (Oracle Java SE Universal Subscription) やポリシー (Oracle Technology Network (OTN) License Agreement for Java SE) に度重なる改訂が行われている。
- MPI Java Binding を利用する使用者を確実に補足・管理することも合わせて検討が必要。

また、OpenJDK も「GPLv2 with the Classpath Exception」というGPL ライセンスの一種が採用されているため、注意が必要である。ただし、こちらはClasspath Exception という例外が設けられており、MPI ライブラリにOpenJDK 自体は組み込まず、実行時にリンクするだけであればGPLv2 の対象外となる。 以上のとおり、Java 言語バインディングはMPI 規格外のOpen MPI 固有仕様のため、今後のユーザ需要や開発コミュニティ側の意向次第では仕様から削除される可能性もある。また、サポートする場合はライセンス管理に特に注意が必要になる。これらを踏まえると、富岳NEXT ではJava 言語バインディングをサポート対象外とするように見直す必要があると考えられる。

3.3.3.1.6.5　他の言語バインディング

- **Python** 言語バインディング
  - Python 言語バインディングは MPI 規格には導入されていない。Open MPI v5.0.0 以降では configure オプション “--enable-python-bindings” が追加されたが、これは PMIx の Python 言語バインディング を有効化するオプションであり、MPI の Python 言語バインディングをサポートするわけではない。 ただし、MPI 規格にはないものの、現在は OSS の MPI for Python (mpi4py) [20] が MPI の Python 言語バインディングとして広く使われており、実質的な業界標準となっている。また、mpi4py は MPI 規格への追従も行っており、2025 年にリリースされたバージョン 4.1.0 では MPI-5.0 規格へのサポー トが追加された。富岳 NEXT においても、Python 言語バインディングとしては mpi4py を利用していくことが想定される。
- **Julia** 言語バインディング
  - Julia は Python のような記述のしやすさと C 言語や Fortran に匹敵する実行速度を特徴としており、近年 HPC 分野でも注目されているプログラミング言語である。Julia 言語バインディングも MPI 規格には導入されていないが、OSS の MPI.jl [17] が Julia 言語バインディングの実質的な業界標準として存在している。富岳 NEXT においても、Julia プログラムから MPI を利用する場合は MPI.jl を用いることになると想定される。

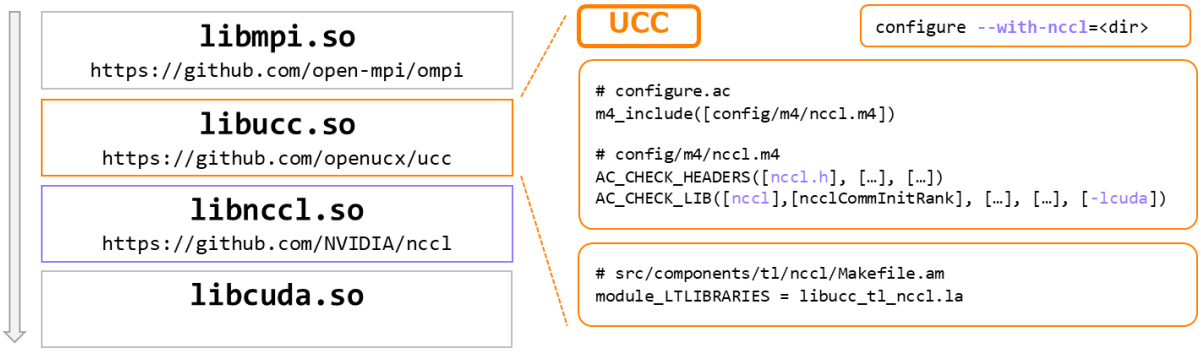
3.3.3.1.7　加速部連携

3.3.3.1.7.1　GPU 間通信ライブラリ連携

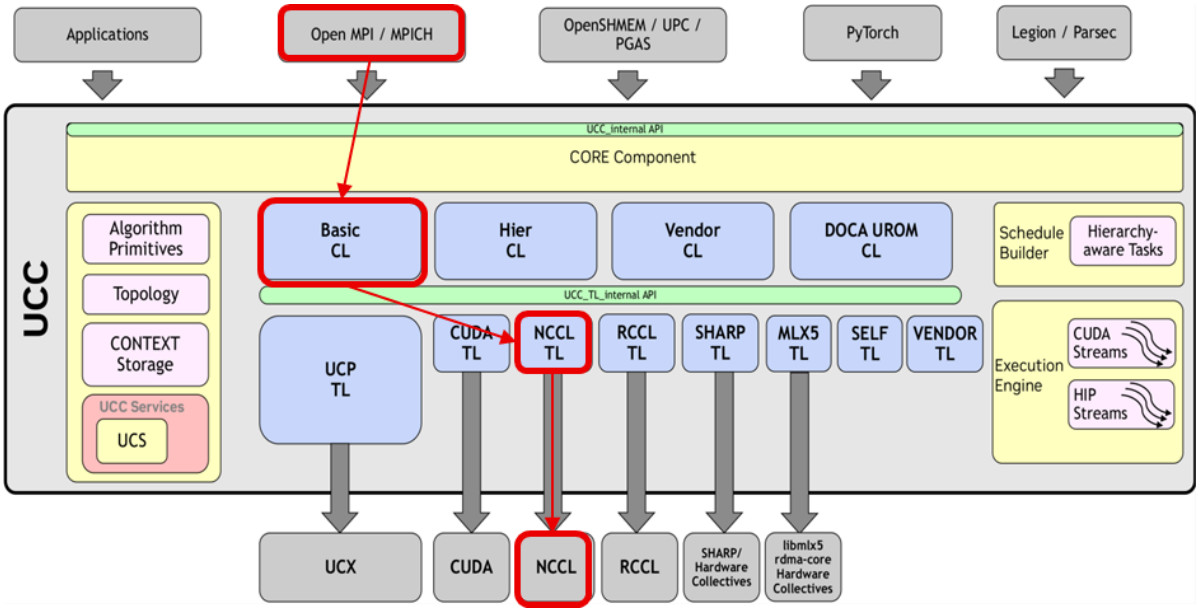
ユーザデータがホストメモリまたは GPU デバイスメモリどちらに存在するかに関係なく、 Allreduce や Alltoall というような集団通信はアプリケーションにとって重要な処理であり続けている。しかしながら、物理的メモリシステムに依存して、メモリ特性やメモリ階層 (内部トポロジ) はまったく異なり、通信アルゴリズムを含む集団通信実装もまったく異なるものとなっており、結果とし て、MPI 実装が提供する従来の汎用集団通信実装を含め、多様な xCCL (Collective Communication Library; 集団通信ライブラリ) が存在するに至っている。MPI 実装側に多様化する集団通信ライブラ リをどう構成し、MPI 実装を介しアプリケーションからどう呼び分けるかは明確でないことが課題である。なお、xCCL はベンダ主導で開発されており、今もなお変化と増殖を繰り返している。 現状では関連 WA / WG で検討中であるが、基本設計では加速部を NVIDIA 系統のGPU を前提として、NCCL [21] が一つのデファクト標準として考え、加速部連携の事例としてNCCL の呼び出し、 NCCL がサポートする集団通信、UCC およびNCCL を有効にして利用する方法を示す。

3.3.3.1.7.2　Open MPI と NCCL の連携

以下にOpen MPI とGPU 間集団通信ライブラリNCCL の連携について示す。



以下にUCC Component Architecture の模式図を示す。



上図での略語はそれぞれ、CL: Collective Layer、TL: Team Layer、NCCL: Optimized primitives for inter-GPU communication（https://github.com/NVIDIA/nccl）である。UCC はGPU 間集団通信ライブラリであるNCCL の呼出しをサポートしている。Open MPI には NCCL の呼び出し実装はなく、UCC を経由して呼ばれる。

3.3.3.1.7.3　NCCL がサポートする集団通信

<coll>	! root	root
allgather	ncclAllGather()	n/a
allgatherv	ncclGroupStart / ncclSend / ncclRecv / ncclGroupEnd	n/a
allreduce	ncclAllReduce()	n/a
alltoall	ncclGroupStart / ncclSend / ncclRecv / ncclGroupEnd	n/a
alltoallv	ncclGroupStart / ncclSend / ncclRecv / ncclGroupEnd	n/a
bcast	ncclBroadcast()	あり
reduce_scatter	ncclReduceScatter()	n/a
reduce_scatter_block	n/a	n/a
reduce	ncclReduce()	あり
barrier	Call allreduce with count = 1	n/a
gather	ncclSend	ncclGroupStart / ncclRecv / ncclGroupEnd
gatherv	ncclSend	ncclGroupStart / ncclRecv / ncclGroupEnd
scatter	ncclRecv	ncclGroupStart / ncclSend / ncclGroupEnd
scatterv	ncclRecv	ncclGroupStart / ncclSend / ncclGroupEnd

上にNCCL がサポートする集団通信ucc_tl_nccl_<coll>_init()を示す。NCCL がサポートする集団通信は8つ(で、それ以外はncclSend/ncclRecv で補完される。

3.3.3.1.7.4　UCC、NCCL を有効にして利用する例

- Open MPI における、UCC 無効の実行方法例

```
mpirun -mca coll_ucc_enable=0 ¥
  osu_allreduce -d cuda -m 1000000000:1000000000
```

- Open MPI における、UCC 有効の実行方法例

```
mpirun -mca coll_ucc_enable=1 -mca coll_ucc_priority=100 ¥
  -x UCC_TL_NCCL_TUNE=allreduce:cuda:0 ¥
  osu_allreduce -d cuda -m 1000000000:1000000000
```

- Open MPI における、UCC 有効の実行方法例

```
mpirun -mca coll_ucc_enable=1  -mca coll_ucc_priority=100 ¥
  -x UCC_TL_NCCL_TUNE=allreduce:cuda:inf ¥
  osu_allreduce -d cuda -m 1000000000:1000000000
```

オプション	概要
coll_ucc_enable	Feature Gate 相当
coll_ucc_priority	UCC 選択優先度
-d cuda	メモリの確保にて cudaMalloc()を呼び出す
UCC_TL_NCCL_TUNE	集団通信個別の優先度

上に UCC および NCCL 関連のオプションを示す。NCCL を使用することで、データが GPU のメモリ上にある場合は、最もデータの移動が少なくなることが期待される。

3.3.3.1.7.5　CPU と GPU 間通信ライブラリ連携

CPU と GPU (加速部) 間のデータ移動と大域同期を軽減することは、大容量データを処理する上での基本事項である。実際、加速部上の演算器を止めないために、データの局所性を含む無駄なデータ 移動を抑制しメモリ帯域の減衰を抑制したり、同期のために演算が待ち状態になることを抑制したり することは、高効率の大量データ処理上有効である。 一般に、集団通信実装は UCX のような低レベルの一対一通信ライブラリの上に、実装することが可能である。GPU-aware MPI [22] の文脈において、こうした一対一通信ライブラリの中の加速部固有トランスポートがどの条件で動作し、加速部の演算器の世代でどのような通信手段と性能特性を有するかは、MPI の集団通信呼出しからは隠蔽され明確でない。 こうした問題に対処するため、今後の加速部のデバイス情報と整合させながら、一対一通信ライブラリの加速部固有トランスポートの特性を見極めることが重要である。現状では関連 WA / WG で検討中であるが、加速部を NVIDIA 系統のGPU を前提とし、GPU-aware 対応の一対一通信ライブラリの選択肢として UCX を仮定し、連携について設調査する必要があると考える。

3.3.3.1.8　ストレージ連携

MPI IO はファイル計算資源の抽象化API(MPI_File_*)に加えて、(1) Data Sieving, (2) Two-Pase I/O, (3) File-system specific Optimization (Lustre では、stripe count, stripe size, stripe offset)の最適化が利用できる。これらは、大量の小さなアクセスを大きな少数アクセスにすることで、ファイルシステム の負荷軽減を支援するものである。 スーパーコンピュータ「富岳」では、富士通のファイルシステム(FEFS, LLIO)向けにMPI も富士通 独自の実装を行ったが、富岳NEXT においては採用するファイルシステムの実装に合わせて利用を検討する。 Open MPI ではMPI IO のコンポーネントとしてompio [23] とROMIO が利用されてきたが、MPICH プロジェクトは、MPI Standard 4.0 で追加されたMPI_Count 拡張に、ROMIO が対応できていないことから、独立のROMIO ライブラリのサポートをやめており、富岳NEXT ではompio コンポーネントを利用すること

が想定される。通信ライブラリとしては、ファイルシステムの選択など、ストレージWG の方針提示を受けてから 詳細設計にて、対応を検討する。以下の節では、ファイルシステムがLustre の場合の連携の例を示す。

3.3.3.1.8.1 MPI IO と Lustre 連携

Lustre [24] はGPLv2.0 ライセンスであり、MPI IO がLustre と連携する場合、Lustre のヘッダを include することで、MPI ライブラリにGPLv2.0 ライセンスが伝播する。GPLv2.0 ライセンスのMPI ライブラリをユーザーアプリが利用しても、GPLv2.0 ライセンスは伝播しないが、MPI ライブラリをアプリと共に持ち出して使用する場合は、アプリケーションへもGPLv2.0 ライセンスが伝播するため、 注意が必要である。Lustre との連携はLustre 提供のヘッダをMPI ライブラリのビルド時にinclude することで、MPI IOと連携できるようにする。

3.3.3.1.9 通信ベンチマークツール連携

プログラミングモデルとしての MPI 実装は各種計算資源を抽象化した API を提供し、通信ライブラリに限らず様々なシステムソフトウェアの集約点となっており、一般的に構成は複雑であり多岐に渡る。また、時代の要請に従い、MPI を構成する個々のシステムソフトウェアのライブラリは中長期的に別の実装に置き換えられたり、廃止されたりする場合もある。短期的には、個々のライブラリはバージョンアップを繰り返し、近年の継続的インテグレーションや継続的デリバリーの深化・普及により、個々のライブラリのリリース間隔は短縮され、3 ヵ月間隔のバージョンアップも珍しくはなくなった。結果として、多くのシステムソフトウェアのライブラリに集約点である MPI 実装は、リリース検証が増加の一途をたどりながらも、適切なリリース管理と品質管理により、今なお成長を続けている。一方で、MPI 実装は数多くの性能に敏感なアプリケーションから使用されており、MPI 実装のバックエンドのライブラリの新バージョンの性能劣化がアプリケーションに波及することもある。各ライブラリからアプリケーションまでの各段で性能検証し早期に性能劣化を検出することで、運用レベルに波及することを防止する必要がある。とりわけ、類似の計算機システムを持たない一品物のスーパーコンピュータシステムでは、カスタムチューニングされたアプリケーションとの組み合わせで、 類似の課題を共有困難な、そのシステムのそのアプリケーションでのみ生じる性能劣化の原因調査に多くのコストがかかっていた。こうした問題を解決するために、継続的ベンチマークと呼ばれるツールが登場している。例えば、Benchpark [25] では、システム仕様、ベンチマーク仕様、および実験仕様（所与のシステム上での特 定のベンチマークの測定仕様）の 3 つのレベルで仕様を記述可能であり、一品物のスーパーコンピュータシステムに対する固有のカスタマイズが可能である。 基本設計では、Benchpark に含まれるベンチマーク GPCNeT / OMB / SMB について比較を行い、 OMB の例を示す。

- GPCNeT (Global Performance and Congestion Network Test)
  - https://github.com/netbench/GPCNET
  - ネットワーク輻輳ベンチマーク
    - network_test: 輻輳なしのベンチ、
    - network_load_test: 輻輳影響のベンチ (ノードの 80 % を congestor)
  - 元々 adaptive routing (スイッチ間の輻輳リンクの自動迂回) 機能の分析用途。
  - アプリ開発者向けのベンチではない。
- OMB (Ohio-state-university Micro Benchmarks)
  - https://mvapich.cse.ohio-state.edu/benchmarks
  - GPU 対応汎用 MPI ベンチマーク (IMB (Intel MPI Benchmarks) replacement)
  - 一対一通信 (10 個)、集団通信 (54 個)、一方向通信 (9 個) の各ベンチ (Host / GPU メモリ対応含む)
- SMB (Sandia MPI micro-Benchmark suite)

- https://github.com/sandialabs/SMB
- より現実的な一対一通信性能のベンチマーク
  - msgrate: キャッシュ効果無効の一対一通信性能ベンチ
  - mpi_overhead: 計算と通信のオーバーラップ環境での一対一通信性能ベンチ

上記 3 つのMPI ベンチマークツールのうち、OMB [26] はMPI 標準の集団通信 (22 個 × 3 タイプ) のうち、MPI_{Scan,Exscan} を除く、ほぼすべてをカバーしている。以下にOMB-7.5.1 MPI ベンチマークプログラム一覧を示す。SMB / GPCNeT は補完的な位置づけであると考える。

collective / API	collective / blocking	collective / non-blocking	collective / persistent	Remarks
MPI_Allgather	osu_allgather	osu_iallgather	osu_allgather_persistent	
MPI_Allgatherv	osu_allgatherv	osu_iallgatherv	osu_allgatherv_persistent	
MPI_Allreduce	osu_allreduce	osu_iallreduce	osu_allreduce_persistent	
MPI_Alltoall	osu_alltoall	osu_ialltoall	osu_alltoall_persistent	
MPI_Alltoallv	osu_alltoallv	osu_ialltoallv	osu_alltoallv_persistent	
MPI_Alltoallw	osu_alltoallw	osu_ialltoallw	osu_alltoallw_persistent	UCC API ×
MPI_Barrier	osu_barrier	osu_ibarrier	osu_barrier_persistent	
MPI_Bcast	osu_bcast	osu_ibcast	osu_bcast_persistent	
MPI_Exscan	(not supported)	(not supported)	(not supported)	UCC API ×
MPI_Gather	osu_gather	osu_igather	osu_gather_persistent	
MPI_Gatherv	osu_gatherv	osu_igatherv	osu_gatherv_persistent	
MPI_Reduce	osu_reduce	osu_ireduce	osu_reduce_persistent	
MPI_Reduce_scatter	osu_reduce_scatter	osu_ireduce_scatter	osu_reduce_scatter_persistent	
MPI_Reduce_scatter_block	osu_reduce_scatter_block	osu_ireduce_scatter_block	(not supported)	
MPI_Scan	(not supported)	(not supported)	(not supported)	UCC API ×
MPI_Scatter	osu_scatter	osu_iscatter	osu_scatter_persistent	
MPI_Scatterv	osu_scatterv	osu_iscatterv	osu_scatterv_persistent	
MPI_Neighbor_allgather	osu_neighbor_allgather	osu_ineighbor_allgather	(not supported)	UCC API ×
MPI_Neighbor_allgatherv	osu_neighbor_allgatherv	osu_ineighbor_allgatherv	(not supported)	UCC API ×
MPI_Neighbor_alltoall	osu_neighbor_alltoall	osu_ineighbor_alltoall	(not supported)	UCC API ×
MPI_Neighbor_alltoallv	osu_neighbor_alltoallv	osu_ineighbor_alltoallv	(not supported)	UCC API ×
MPI_Neighbor_alltoallw	osu_neighbor_alltoallw	osu_ineighbor_alltoallw	(not supported)	UCC API ×
22 関数	*20 ベンチ*	*20 ベンチ*	*14 ベンチ*	

point-to-point	one-sided
osu_bibw	osu_acc_latency
osu_bw	osu_cas_latency
osu_latecy	osu_fop_latency
osu_latency_mp	osu_get_acc_latency
osu_latency_mt	osu_get_bw
osu_mbw_mr	osu_get_latency
osu_multi_lat	osu_put_bibw
osu_bibw_persistent	osu_put_bw
osu_bw_persistent	osu_put_latency
osu_latency_persistent	
10 ベンチ	*9 ベンチ*

3.3.3.1.9.1 汎用通信ベンチマーク

現時点で、通信ベンチマークのソフトウェアとして著名な通信ベンチマークの一つは Ohio-state-university Micro Benchmarks (OMB) [26] である。OMB では、MPI の一対一通信、一方向通信、および集団通信の各操作に対し、C 言語呼出しプログラムだけでもそれぞれ、10 ベンチマーク、9 ベンチマーク、54 ベンチマークのプログラムが存在する。各ベンチマークは、GPU デバイスメモリ上のユーザ通信バッファを指定する、いわゆる GPU-aware MPI 向けのオプションも拡張されてい る。また、OMB はBenchpark にもベンチマークに含まれ、標準的な通信系ベンチマークの位置を占めつつある。以下にOMB の使用例を示す。

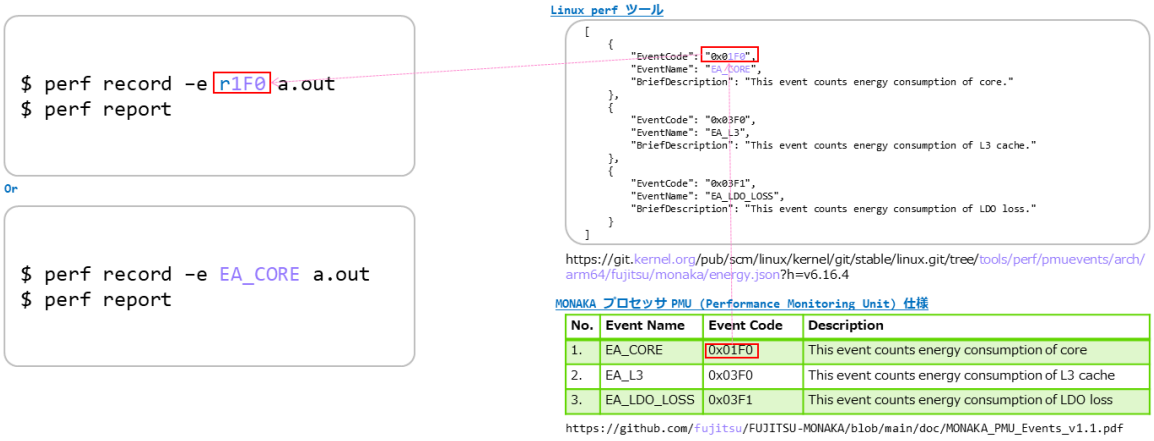
```
$ mpirun -n 2 osu_reduce --iterations=100 --warmup=15 ¥
    --message-size=$((32*1024*1024)):$((32*1024*1024)) --type mpi_float

# OSU MPI Reduce Latency Test v7.5
# Datatype: MPI_FLOAT.
# Size       Avg Latency(us)
33554432            11466.67


$ mpirun -n 2 osu_reduce --iterations=100 --warmup=15 ¥
    --message-size=$((32*1024*1024)):$((32*1024*1024)) --type mpi_float --full

# OSU MPI Reduce Latency Test v7.5
# Datatype: MPI_FLOAT.
# Size       Avg Latency(us)   Min Latency(us)   Max Latency(us)  Iterations
33554432            11252.42          11209.48          11295.35         100
```

OMB では、各プロセスにおいて、集団通信を iterations (iters) 回実行して、通信の総経過時間 (timer) を集計し、iteration 当たりの(平均)時間 (latency) を求める。 最後に、MPI_Reduce with MPI_{MIN,MAX,SUM} を使って、プロセス間の最小・最大・平均 (avg_time) を表示する。OMB は、継続的に拡張が続けられており、進化・対応も速く、非常に大きなスイートとなっている。

3.3.3.1.10 細粒度電力量可視化

計算ノードの電力量を取得する場合一般的には、システムボード上に配置された電力センサに ACPI 経由でアクセスする OS ドライバ (例えば acpi_power_meter) を使用する。しかしながら、ボード単 位または (複数のセンサを具備する場合) ソケット単位での値となり、CPU コア単位等の粒度では計測できない。このため一般的には、CPU の Performance Monitoring Unit (PMU) イベントを使い各コ アの使用電力量を推定する手法が用いられる。また、電力センサの更新周期がミリ秒程度のものが多く流通しており、例えば数十～数百マイクロ秒の関数単位の電力量計測では精度上の問題も存在する。 アプリケーションの演算カーネル単体の評価等、ごく限られた条件でしか電力量を測定できなかった。 A64FX では PMU イベントが拡張されており、A64FX 固有の Energy Analyzer (EA) イベントが 追加されている。EA はコア単位で電力量 (nJ) を計測可能である。A64FX 後継プロセッサにおいても EA がサポートされる予定である [27]。しかしながら、EA カウンタ 1 刻み当たりの電力量 (nJ) の 確定は 2027 年以降の予定である。

以下にperf コマンドの利用例を示す。



以下にFUJITSU-MONAKA プロセッサ PMU の消費電力に関する性能カウンタを示す。 FUJITSU-MONAKA プロセッサ PMU には消費電力に関する性能カウンタが7 つあり、perf コマン ドに指定することで、電力可視化ができる。

No.	Event Name	Event Code	Description
1.	EA_CORE	0x01F0	This event counts energy consumption of core
2.	EA_L3	0x03F0	This event counts energy consumption of L3 cache
3.	EA_LDO_LOSS	0x03F1	This event counts energy consumption of LDO loss
4.	EA_MAC	0x0080	This event counts energy consumption of MAC
5.	EA_MEMORY	0x0090	This event counts energy consumption of memory
6.	EA_HA	0x00A0	This event counts energy consumption of HA
7.	EA_PCI (un-core)	0x0080	This event counts energy consumption of PCI

3.3.3.1.11 詳細設計における検討内容と方針

項番	項目	検討方針
1	システムおよびハード依存部に関する通信ライブラリの開発項目抽出	富岳 NEXT システムのアーキテクチャ確定に伴い、通信ライブラリで開発すべき項目を抽出。システムネットワーク、インターコネクト、トポロジなどアーキテクチャが確定し次第、通信ライブラリとして性能観点など開発すべきアーキテクチャ固有事項を検討する。
2	技術動向調査の継続	本報告書に記載した技術動向調査を継続調査し、開発すべき項目を抽出。
3	通信ライブラリ WG の TWI に関する通信ライブラリの開発項目抽出	通信ライブラリ WG の TWI であるベンチマークソフト整備におけるアプリケーション評価などから開発すべき項目を抽出。

項番1 については、詳細設計以降にスケールアウトネットワーク、スイッチ、トポロジが決定したタイミングで、検討項目内容を調整する。項番2 について、OSS のバグ修正や性能改善が発生する可能性の高い項目を上げて調査・開発すべき項目を抽出し、開発優先度を検討、優先度の高い項目から必要な開発を行う。項番3 についても通信ライブラリWG で議論の上、調査を継続する。

3.3.3.2 加速部

3.3.3.2.1 加速部に係る通信ソフトウェア

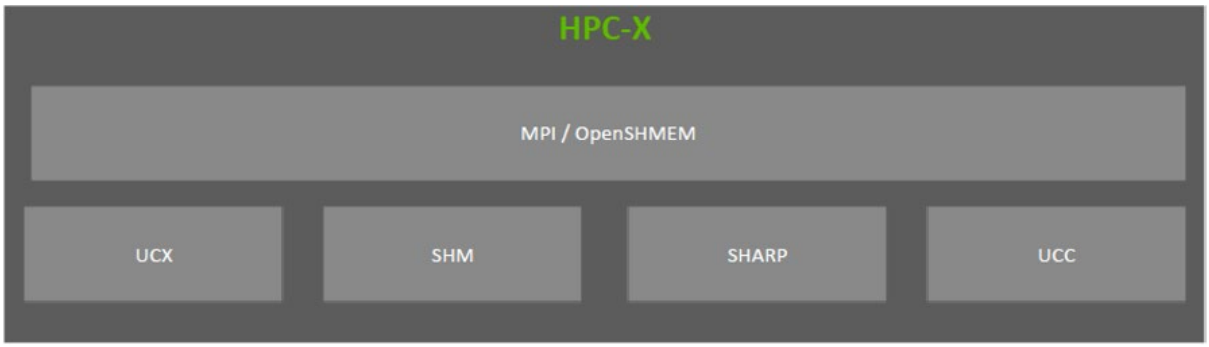
- MPI/OPENSHMEM：従来型の HPC アプリケーションをサポートする
- NCCL：すべての AI ユースケースおよび一部の HPC ユースケースをサポートする。これには集団通信、1 対 1 通信、および GPU 主導の片方向通信が含まれる。
- NVSHMEM：GPU 主導の通信を提供し、片方向通信および集団通信を含む。
- AI ワークロードにおいては、Mixture-of-Experts（MoE）並列化で広く利用されている。
- HPC 分野では、Gromacs や数値計算ライブラリなどのアプリケーションで利用されている。
- NIXL：メモリおよびストレージ向けの汎用 I/O 転送ライブラリである。
- UCC：MPI および OpenSHMEM の双方に対して HPC 向け集団演算を提供する。
- UCX：MPI、OpenSHMEM、NIXL などのライブラリが NVIDIA ネットワーキング上で動作することを可能とする、汎用の低レベル通信ライブラリである。

3.3.3.2.2 加速部に係る技術作業項目（TWI: Technical work item）

- ライブラリおよびドライバのサポート: NCCL、UCX、MPI、GPU ドライバについて、安定かつ最適化されたバージョンを提供するとともに、テレメトリ取得用フックを提供する。
- ベンチマークツールの拡張: ベンダ提供のベンチマークツールに対し、新しい NCCL COLLECTIVE や UCX トランスポートオプションなどの機能拡張を行う。
- 性能カウンタの公開: NIC／GPU／CPU のテレメトリ情報を公開し、ベンチマークワークフローへの統合を可能にする。
- トラブルシューティングにおける協業: ファームウェア、ドライバ、通信ライブラリに起因する 可能性のあるベンチマーク異常について、共同で解析・対応を行う。
- ロードマップの整合: 将来のライブラリリリース、オフロード機能、適応型ルーティング機構に関する更新情報を共有し、ベンチマーク対象範囲と整合を取る。

3.3.3.2.3　HPC-X

HPC-X は、NVIDIA が提供する HPC 向け通信ソフトウェアパッケージであり、MPI および OpenSHMEM の通信フレームワークを含む。HPC-X には、MPI と OpenSHMEM の実装が含まれており、これらは NVIDIA の高性能 RDMA 通信ミドルウェア上に構築されている。HPC-X に含まれるMPI およびOpenSHMEM の実装では、1 対1 通信、RMA（Remote Memory Access）、およびコレクティブ通信が、UCX、UCCおよびSHARPを用いて実装されている。これらの通信ミドルウェアは、ネットワークオフロード機能およびアクセラレータを透過的に活用するよう設計されている。



HPC-X の主要な役割の一つは、MPI のセマンティクスおよび可搬性を維持することである。MPI は膨大な既存アプリケーションコードを有する標準規格である。富岳NEXTにおいては、これら既存コードを自然に実行可能としつつ、高性能を実現することが求められる。HPC-X は、UCX を通じて最新のネットワーキング機能およびGPU 通信機能を取り込みつつ、MPI-3 標準に準拠した安定的な実装を提供することにより、これを可能としている。HPC-XにおけるGPUDirectサポートは、多くのGPU対応アプリケーションにとって不可欠であり、CPU 専用コードをGPUへ段階的に移行する上でも必要である。

一方で、HPC-Xは万能な解決策ではない。最高水準の集団通信性能、CUDA ストリームの認識、またはGPU 起点の通信(GPU-initialted communication)を必要とするAI ワークロードに対しては、NCCL またはNVSHMEMが必要となる。したがって、富岳NEXTにおいては、HPC-XがHPCアプリケーションの基盤を担い、AI指向の通信は主としてNCCL をはじめとする他のライブラリに委ねられる。これらのライブラリは相互に互換性を有しており、双方の機能を必要とするアプリケーション、またはそれらに依存するライブラリを利用するアプリケーションにおいて併用可能である。

ベンチマークの観点では、HPC-X の評価軸はNCCL とは異なる。HPC-X においては、MPI ベースの1 対1 通信、集団通信、RMA 性能、およびGPU 対応MPI としてのGPU メモリ通信性能が主要な評価対象となる。ピーク性能のみならず、ノード数増加に伴うスケーラビリティ、多様な通信パターンに対する安定性、ならびにネットワークオフロードの有無による差異を評価する必要がある。これらの観点を反映するため、従来のMPI ベンチマーク（例：OMB）を富岳NEXT の実構成に即して拡張することが求められる。一方、NCCL は独自の集団通信ベンチマークを提供している。

TWIの観点からは、HPC-Xは自動化された通信ベンチマーク基盤を用いて継続的に評価すべき対象である。基盤ライブラリとして、その性能および安定性はシステム全体の信頼性に直結する。そのため、構成変更やライブラリアップデートのたびに手動評価に依存するのではなく、性能劣化や異常を早期に検出可能とする自動化ベンチマーク枠組みが必要である。

設計上の検討事項としては、富岳NEXTにおいてどのMPI実装を標準とするか、ならびにHPC-Xのサポート範囲をどのように定義するかが挙げられる。HPC-X をデフォルトMPI 実装として採用する場合、その更新方針、他の

MPI 実装との共存方法、およびアプリケーションへの影響を明確に定義する必要がある。また、OpenSHMEM をどの程度利用およびサポートするかも設計上の論点である。これらの課題については、詳細設計フェーズにおいて理研、NVIDIA および富士通社の間で継続的に議論する。

3.3.3.2.3.1 NCCL

NCCL（NVIDIA Collective Communications Library）は、NVIDIA が提供するGPU間通信ライブラリであり、当初は集団通信および1 対1 受信機能を主対象として開発されたものであり、AIフレームワーク、GPUハードウェア、クラウドサービス、AI企業など、多くのソフトウェアやハードウェア、サービス、組織からなる大規模なエコシステムにより支えられている。最新バージョンであるNCCL 2.29 では、ホスト起点の通信およびデバイス起点の通信が追加された。これは、DeepEP などのAI ワークロードにおけるNVSHMEM の成功を背景としたものであり、AIアプリケーションがあらゆる種類の通信を単一の通信ライブラリで実行可能とすることを目的としている。

- NCCL が提供する通信処理は以下である。
  - ✧ 集団通信：AllReduce、Reduce、ReduceScatter、Broadcast、Gather、Scatter、AllGather、AllToAll
  - ✧ 1 対 1 通信（双方向およびワンサイド）：Send／Recv、Put／PutSignal／WaitSignal
  - ✧ 対称メモリ（NVL 対応）：ロード／ストアアクセス、マルチキャストポインタ、バリア
  - ✧ デバイス機能：GPU 起点のネットワーキング（GIN）：put、flush、signal、barrier

  これらの通信操作は GPU メモリ上で動作し、CUDA ドライバおよびランタイムと連携して機能する。NCCL は、PCIe、NVLink、InfiniBand／RoCE、ならびにサードパーティ製ネットワークを含む NVIDIA GPU インターコネクト上で動作する。NCCL は、HPC アプリケーションおよびライブラリに加え、PyTorch、JAX、NeMo などの深層学習フレ ームワークから直接利用されている。NCCL は深層学習フレームワークと CUDA の間に位置し、GPU 間通信を担う。
- NCCL は以下の機能を含む。
  - ✧ トポロジ検出
  - ✧ 通信経路探索
  - ✧ 性能チューニング
  - ✧ バッファ登録
  - ✧ CUDA カーネルおよび各種関数
  - ✧ 通信トランスポート（共有メモリ、CUDA ピアツーピア、ネットワーク）

  NCCL は、ノード内およびノード間の GPU 通信の双方をサポートする。ノード内では NVLink および PCIe を利用し、NVLink ドメイン間では InfiniBand や RoCE などの RDMA 対応ネットワークを活用する。NCCL はまた、SHARP などのネットワークオフロード機構も利用可能である。
- NCCL ベンチマークの検討

  ベンチマークの観点からは、AllReduce や Alltoall など主要な集団演算の性能、GPU 数増加に伴うスケーラビリティ、NVLink 世代間の差異、ならびにネットワークオフロードの有無による挙動差が重要な評価軸となる。さらに、実アプリケーションで用いられる通信パターンと、nccl-tests のような合成ベンチマークで表現される通信パターンとの間に存在する乖離をどのように扱うかも、重要な運用上の検討事項である。これらの論点については、詳細設計フェーズにおいてさらに詳細に議論される予定である。

3.3.3.2.3.2 NVSHMEM

NVSHMEM は、NVIDIA が提供するGPU クラスタ向け通信ライブラリであり、片方向通信および集団通信を

主対象とする。NVSHMEM は、GPU 起点の通信（GPU-initiated communication）という概念を先駆的に導入した。この機構により、ホスト起点通信では実現できない水準の計算と通信の融合が可能となる。MPI や OpenSHMEM では、通信操作はCPU 主導またはフレームワーク主導で発行されるのに対し、NVSHMEM の重要な特徴は、GPU カーネル自体が直接通信操作を発行できる点にある。この概念の有効性はNVSHMEM において実証されており、前述のとおり、近年ではNCCL の機能にも取り込まれている。本設計は、計算と通信の融合（compute-communication fusion）を可能とするものであり、GPU カーネル内で生成された計算結果を、CPUの明示的な介在なしに直ちに他のGPUへ転送することを可能とする。その結果、レイテンシおよびオーバーヘッドの低減が期待される。NVSHMEMは片方向通信モデルを採用しており、MPIで一般的に実装される双方向通信モデルとは本質的に異なる。このため、不規則な通信パターンや多数の小規模メッセージを伴うワークロードに対して、NVSHMEM はより自然に適合する。したがって、NVSHMEM が不規則通信やMoE などの通信集約型AIワークロードにおいて強みを示すことは、富岳NEXTで想定されるワークロード特性とも整合すると考えられる。

- NVSHMEM が提供する通信モデル
  - ✧ CPU 起点および GPU 起点の片方向通信
  - ✧ CPU 起点および GPU 起点の集団通信

  NVSHMEM は、MPI および OpenSHMEM と相互運用可能であり、MPI 通信と NVSHMEM 通信を組み合わせたハイブリッド利用が可能である。NVSHMEM の API はホスト API とデバイス API に分かれており、後者は、GPU カーネルから直接呼び出すことが可能である。これらの API により、GPU メモリ領域間における put 操作および集団通信操作が実行可能となる。NVSHMEM は AI および HPC の双方のワークロードで利用されている。
- 代表的な例は以下のとおりである。
  - ✧ Mixture-of-Experts（MoE）を用いた AI 学習および推論
  - ✧ GROMACS
  - ✧ cuFFTMp
  - ✧ QUDA
  - ✧ LBANN

  NVSHMEM は、従来 NCCL では表現が困難であった通信パターンをサポートするライブラリとして重要な位置を占めてきた。しかしながら、現在では NCCL もデバイス起点通信および片方向通信機能の追加を開始しており、NVSHMEM と NCCL の差異は、今後は簡潔性、柔軟性、拡張性といった他の特性へと軸足が移ると考えられる。NVSHMEM のモデルは非常にクリーンかつシンプルであるが、複数コミュニケータのサポートや、デフォルトコミュニケータのサイズ変更機能は備えていない。一方で、集団通信および片方向の 1 対 1 通信を含む多数の通信関数を有しており、これらはいずれもホストおよびデバイスの双方から利用可能である。また、ユーザが持つ操作イメージは環境によらず統一されており、プログラマは NVLink ドメインとネットワークの違いを意識する必要がない。NVSHMEM は、MPI とも大きく異なる特性を有しており、その違いは多くの場合、NCCL との違いと同様である。したがって、すべてのアプリケーションが NVSHMEM に適しているわけではなく、NVSHMEM は MPI／OpenSHMEM（HPC-X）および NCCL と併用されることが想定される。
- ベンチマークの観点

  ホスト起点の通信については、NVSHMEM と他の通信 API を直接比較評価することが可能である。一方、デバイス起点の通信については、客観的な比較指標を定義することが困難である。GPU カーネル内部から発行される NVSHMEM 通信については、通信性能を単独で評価するだけでなく、計算と通信を

組み合わせたエンドツーエンド性能を評価する必要がある。具体的なベンチマーク手法については、詳細設計フェーズにおいて協議して定義する予定である。

3.3.3.2.3.3 NXIL

NIXL（NVIDIA I/O eXchange Library）は、NVIDIA が提案する通信およびデータ移動スタックにおいて、MPI、NCCL、NVSHMEM などの通信ライブラリとは異なる役割を担う汎用データ転送ライブラリである。NIXL は、MPI のようにコレクティブ通信を含む完全な通信セマンティクス一式を提供するものではない。代わりに、メモリおよびストレージを含む多様なデータドメイン間の転送を、統一的な抽象化の下で処理することを目的として設計されている。

NIXL の設計は、層構造の明確な分離を採用しており、上位層にはNorth-Bound NIXL API、下位層にはSouth-Bound Backend API および複数の転送バックエンドが配置される。アプリケーションおよび上位ミドルウェアは、送信元、送信先、データサイズといった論理情報のみを指定し、実際の転送経路および転送機構の選択はNIXL内部に委ねられる。想定される下位層の転送バックエンドには、UCX（またはlibfabric）、POSIX、GPUDirect Storage（GDS）、GPU-NetIO、ならびにストレージシステムおよびオブジェクトストア向けのカスタムバックエンドが含まれる。NIXL はこれらのバックエンドをプラグイン型アーキテクチャにより統合する。富岳NEXT の文脈においては、NIXL はDynamo やSGLang など、本システム上にユーザが展開する可能性のある推論サービングフレームワークと関連する。特に、メモリおよびストレージ階層のサポートが重要となるKV キャッシュのようなAI パターンを実装する上で、有効な手段となる。

ベンチマークの観点では、NIXLはNCCLやMPI と同一の指標で評価可能な対象ではない。NCCLやMPI が主として通信帯域、レイテンシ、スケーラビリティの観点で評価されるのに対し、NIXL の主要な評価軸は、メモリおよびストレージドメインをまたぐ複雑なデータ移動の挙動にある。具体的には、GPU メモリとCPU メモリ間の転送スループット、GPU メモリとNVMe 間のスループット、ノード内およびノード間をまたぐ転送のオーバーヘッド、repost 利用時の性能安定性、大規模同時転送時の輻輳および競合の影響、長時間実行時における性能ドリフトやリソースリークの有無などが重要な検討事項となる。これらの特性は単発のマイクロベンチマークでは把握できず、自動化された長時間・多構成評価が必要である。この観点から、自動化ベンチマーク基盤および可視化・長期トレンド分析の枠組みは、NIXLが実現するエンドツーエンド性能を捉える推論サービングフレームワーク（例：Dynamo）を主対象とすることを検討すべきである。

設計上の検討事項としては、富岳NEXT においてNIXL をどのように利用するか、あるいは利用するか否かを明確に定義する必要がある。NIXL を利用する推論サービングフレームワークを展開する場合には、NIXLの実装方針を明確化する必要がある。低レベルネットワークスタックにUCXが含まれる場合には、NIXL の動作原理は比較的明確である。

3.3.3.3 高度化

富岳NEXTにおける通信ライブラリの設計に関して、詳細設計に先立ち整理すべき基本的な考え方および設計上の留意点をまとめる。通信ライブラリは、システム全体の構成、実行環境、運用条件と密接に関係する基盤的ソフトウェア層であり、特定のハードウェア構成や利用形態に強く依存した設計を初期段階で固定することは、将来的な性能発揮や拡張性の観点から適切でない。 通信ライブラリに関する技術動向、ならびにシステム全体、CPU、GPU の各視点からの整理については、前節また、別紙（富士通社およびNVIDIA社の基本設計の報告書）にて整理されており、ここではそれらを前提とする。ここでは、特定の通信ライブラリ構成や実装方式を確定することを目的とせず、将来確定するハードウェア構成および実行環境において通信性能を適切に引き出すための設計思想、

判断軸、および設計上の留意点を整理することに主眼を置く。具体的には、通信性能の把握および問題切り分けの考え方、ならびに実行環境・運用条件を踏まえた通信設計上の留意点を整理し、詳細設計フェーズにおける検討の指針を示すことを目的とする。

3.3.3.3.1　通信性能の把握および問題切り分けに関する考え方

通信性能は、通信ライブラリの実装のみならず、システム構成、実行環境、運用条件など複数の要因が重なって決定される。このため。性能低下やばらつきが観測された場合に、単一の要因に起因すると仮定するのではなく、段階的に要因を切り分けて把握するための枠組みをあらかじめ設計に組み込んでおくことが重要となる。 本設計では、通信性能の評価を、通信ライブラリ単体の性能を決定づけるものではなく、システム全体の挙動を理解し、問題の所在を整理するための設計上の手段として位置づける。実アプリケーション評価、通信プロファイリング、通信ベンチマークは、それぞれ異なる観点から情報を提供するものであり、いずれか一つに依存するのではなく、相補的に用いることを基本とする。 また、通信性能の把握および問題切り分けは一時的な作業ではなく、詳細設計および運用フェーズにおいても継続的に実施されることが望ましい。このため、本設計では、通信ライブラリに関するベンチマークについて、単発の評価用途に留まらず、継続的な実行および結果比較が可能な形で利用されることを前提とする。 この考えに基づき、基本設計段階では、GH200 環境において Benchpark を用い、OMB（OSU Micro-Benchmarks）等の代表的な通信ベンチマークを実行可能な形で整備し、通信ライブラリの基本的な挙動を同一条件下で継続的に確認できる環境を構築した。今後も、富岳NEXT向けの性能評価機器等において、同様の通信ベンチマーク環境を段階的に整備していくことを想定する。 これらの通信ベンチマークは、通信性能を単独で評価することを目的とするものではなく、システム構成、実行環境、運用条件の違いが通信挙動に与える影響を把握するための補助的な手段として位置づける。継続的な通信ベンチマークの活用により、性能変化の傾向把握や、問題発生時の初期切り分けを容易にし、詳細設計および運用フェーズにおける検討に資することを想定する。

3.3.3.3.2　実行環境・運用条件を考慮した通信設計上の留意点

通信性能は、ジョブ配置方針、co-allocation、 OS ノイズ、メモリ管理方式といった運用条件の影響を受ける。このため、通信ライブラリの設計においては、 実行環境が通信にとって常に最適な状態であることを前提としない考え方が求められる。 本設計では、通信処理に関して、実行環境依存性の高い要素を意識的に切り分け、その影響が通信機能全体に波及しないよう配慮することを基本とする。一方で、ストレージ構成、ジョブスケジューラの挙動、OSの詳細な動作といった通信ライブラリ単体で制御することが困難な事項については、通信ライブラリ単体での解決を前提とせず、運用や上位層との役割分担に委ねる。 また、実行環境や運用条件は、システム導入時点で固定されるものではなく、利用形態の変化や運用方針の見直しに伴い、時間とともに変化する可能性がある。通信設計においては、こうした変化を前提とし、特定の運用条件に強く依存しない構成とすることが望ましい。 特に、ジョブ配置方針や資源共有のあり方は、通信性能のばらつきに影響を与える要因となり得る。本設計では、これらの要因が通信ライブラリの基本動作に直接的な制約を与えないよう、設定値や運用調整、上位層との協調によって吸収されることを前提とした設計上の留意点として整理する。

3.3.3.3.3　通信設計に関する引き継ぎ事項（詳細設計フェーズに向けて）

通信に関する基本的な考え方および設計上の留意点を整理し、詳細設計フェーズにおける検討に向けた前提条件を示した。通信ライブラリに関する具体的な構成、ハードウェア特性に即した最適化手法、ならびに実装方針については、ハードウェア構成および実行環境の詳細が確定した後に検討される。

詳細設計フェーズにおいては、本章で整理した以下の考え方を前提条件として検討を進めるものとする。

- 通信性能は単一要因で決定されるものではなく、段階的な把握および問題切り分けを前提とすること

- 通信ベンチマークは性能評価の目的ではなく、挙動理解および切り分けのための設計上の手段として位置づけること
- 実行環境・運用条件に起因する性能変動を、通信ライブラリの基本動作から切り離す設計思想を維持すること

これらを踏まえ、計算、メモリ、I/O、運用といった他の設計要素との整合性を考慮しながら、段階的に詳細設計を進めるものとする。

なお、本通信ライブラリは、並列アプリケーションに対して通信機能を提供する基盤ソフトウェア層として、通信方式や実行環境の差異を考慮しつつ、将来確定するハードウェア構成において通信性能を適切に発揮できるような構造を提供することを目的とする。本基本設計フェーズにおいては、特定のハードウェア構成や実行条件を前提とした数値的な通信性能（レイテンシ、帯域等）を定めることは行わず、これらの性能目標については、ハードウェア構成、実行環境、通信パターン等の条件を明確にしたうえで、詳細設計フェーズ以降に検討・設定され、その条件下において想定した性能が得られていることを確認することを想定する。

3.3.4 AI ソフトウェア

3.3.4.1 全体システム・CPU 部

3.3.4.1.1 本文書の位置付け

本文章は、富岳NEXTプロジェクトにおいて富士通の責任範囲となるCPU部について、AI用途におけるフレームワークおよびライブラリのソフトスタックに関する基本設計をまとめたものである。

フレームワーク、ライブラリ以外のレイヤについては、以下のように扱う。

- 上位のアプリケーションレイヤについては、コデザイン検討会のスコープであるが、本文書のソフトスタックが想定するユースケースを明確にするため、CPU の利用が見込まれるケースを本章の 3.3.4.1.2 にも記載する。
- 下位のシステムソフトレイヤ（ジョブ管理、コンテナ管理、OS 等）については、運用システム・利用環境整備検討会のスコープであり、本文書のソフトスタックとは独立したテーマであるため、本文書では取り扱わないものとする。

3.3.4.1.2 想定ユースケース

本節では、AI処理におけるCPUのユースケース想定を示す。AI処理は、大きく分けて学習と推論に分けられる。学習における想定を3.3.4.1.2.1に、推論における想定を3.3.4.1.2.2に、それぞれ示す。また、富岳NEXT特有のユースケースであるAIとHPCの融合について、3.3.4.1.2.3に想定を示す。ファインチューニングについては、考慮すべき事項が学習と同様であるため、3.3.4.1.2.1で併せて示す。

3.3.4.1.2.1 AI モデル学習・ファインチューニング（国産生成 AI モデル）

学習処理は、大規模な学習データセットをバッチ処理する特徴から高い演算性能を要求される処理であり、一般的にはGPU等のアクセラレータが用いられる。しかしながら、学習処理の周辺処理はCPUが担っており、CPU性能が学習処理のボトルネックとなるケースが存在する。

本文書では、富岳NEXTの目的から、特に以下をユースケースとして想定した。

- 強化学習の報酬計算
  - 強化学習の報酬としてシミュレータや探索等の処理（例： [1]）を CPU で実行し、その性能が学習処理を律速しているケース。具体的なアプリケーションとしては、SmartSim [2]等が挙げられる。
- Surrogate Model の学習
  - AI 学習処理を GPU が、HPC シミュレーションを CPU が担う。
  - HPC シミュレーションの演算量が学習処理よりも大きい場合、逆の分担もありうる。
- その他、3.3.4.1.2.2 に記載のユースケース。
  - 学習においても同様のことが言える。 （軽量モデルのファインチューニング、学習データの前後処理、等）

3.3.4.1.2.2 AI 推論

推論処理は、学習と比べて演算規模が小さいため、用途や性能指標次第ではCPUがGPUを上回ることや、GPUに加えてCPUも活用することでシステム全体の演算効率が向上する場合があることが知られている。

本文書では、特に以下をユースケースとして想定した。

- CPU が GPU を上回るケース

- GPU で動作させるためのオーバーヘッド（データ転送、Kernel 起動など）と比較して演算量が小さく、GPU で動作させると効率が悪いもの （例：小バッチサイズ推論 [3]、SLM、言語モデル以外の軽量モデル）
- 分岐処理が多いなど、GPU のアーキテクチャに適していないもの （例：random-forest）

- CPU 活用によりシステム全体の性能が向上するケース
  - システム内に規模が異なる複数のモデルが混在し、小規模なものを CPU が担うもの （例：RAG の Embedding Model、Speculative Decoding の draft-model）
  - 推論処理の一部を CPU が担うことによって性能向上を図れるもの （例：LLM の Decode 処理、MoE,MoA [4]の GPU メモリに載らない Agent,Expert）
  - システム内に AI 以外の高負荷処理が混在し、CPU と GPU で分担するもの （例：推論の前処理としてビッグデータ解析が必要な場合）

3.3.4.1.2.3 HPC と AI の融合

富岳NEXTはコンセプトの一つとしてAI for Scienceを掲げており、HPCとAIの融合は重要テーマの一つである。具体的にHPCとAIがそれぞれどのような役割を担うかはシステム要件毎に多様だが、本文書では以下をユースケースとして想定した。

- AI for HPC application running：HPC 計算のために入力データ、走行条件などを AI が生成する。
- AI for HPC application programming：HPC アプリケーションのプログラミングを AI が支援する。
- AI Surrogate for HPC：HPC 計算を AI 処理によって代替する。
- HPC for AI training：AI の学習データを HPC 計算によって生成する。
- HPC for AI agentic workflows：AI-Agent が MCP サーバ等を介して動的に HPC 計算タスクを生成し起動する。

3.3.4.1.3 AI ソフトスタック

本節では、3.3.4.1.2のユースケースを実現する具体的なソフトスタックを示す。CPUのソフトスタックはOSSのみから構成される。GPUのソフトスタックについては、NVIDIA社の責任範囲であるため、本文書のスコープ外とする。但し、CPUとGPUの協調動作を実現するソフトウェア群については、CPUコデザインの観点でも重要であるため本文書でも取り扱うこととする。それらのソフトウェアについては、一部にNVIDIA社のOpen Binaryを含んでいる。

ソフトスタックは表形式で記載しており、表は以下の要素から構成される。

- ソフト名：ソフトウェアの名称。
- 用途：ソフトウェアの用途。Training, Inference, Fine-Tuning, MLOps, 等。
- レイヤ：ソフトスタックにおける階層。
- デバイス：CPU, CPU/GPU, CPU&GPU のいずれか。
  - CPU：CPU 上で AI 処理を行う際に使用するソフトウェア。
  - CPU/GPU：CPU 上で AI 処理を行う際に使用するソフトウェアだが、GPU サポートがあり、GPU 上の AI 処理においても同様の使い方ができるもの。
  - CPU&GPU：CPU と GPU の協調動作を行う際に使用するソフトウェア。
- 機能：ソフトウェアの主たる機能。
- Arm 動作：◎, ○, △のいずれか。

- ◎：富士通が過去に当該ソフトウェアを Arm 機上で動作させた実績があるもの。※ 全機能の検証ではなく、1 パターン以上の何らかの入力に対して正常動作したもの。また、性能については考慮せず動作の有無のみ記載。
- ○：富士通での動作実績は無いが、ソフトウェア開発元（OSS コミュニティ等）が Arm 機 サポートを表明しているもの。
- △：Arm 機上で動作するか現時点では不明のもの。これらのソフトウェアは、詳細設計のフェーズで Arm 機上での動作確認を実施し、 動作しない場合には同等機能を有する他ソフトウェアでの代替を検討する。

なお、本文書のソフトスタックは2026年1月現在において主流のOSSから構成されており、富岳NEXTの稼働・運用時点では同等機能を有する他のソフトウェアに置き換わる可能性がある。また、ソフトウェアバージョンについても、アップデートに追従して常に最新のものを利用することが適切であることから、本文書中では明記していない。稼働・運用時点で必要なOSSが富岳NEXTのCPUをサポートしていない場合や、CPUの機能を十分に活用できていない場合には、OSSコミュニティにissueやpull requestの形で働き掛け、機能・性能の改善を図る必要がある。

3.3.4.1.3.1　機械学習向けソフトウェア

本項では、主な用途が機械学習であるソフトウェアを示す。機械学習向けには一般的に利用されるソフトウェア（Scikit-Learn, NumPy, Pandas等）に加え、大規模な計算資源を効率的に扱うために、分散並列のためのソフトウェアとしてDASKやModinなども挙げている。

ソフト名	用途	レイヤ	デバイス	機能	ARM 動作
XGBoost	Data Analysis	Framework	CPU	勾配ブースティング決定木の実装	◎
Scikit-Learn	Data Analysis	Framework	CPU	機械学習向けツール・アルゴリズム群	◎
Statsmodels	Data Analysis	Framework	CPU	統計モデリングと検定、データ探索のためのPythonライブラリ	○
NumPy	Data Analysis	Library	CPU	数値計算のためのPythonライブラリ	◎
oneDAL	Data Analysis	Library	CPU	データ分析・機械学習のための最適化ライブラリ	◎
SciPy	Data Analysis	Library	CPU	科学計算のためのPythonライブラリ	◎
DASK	Data Analysis	Parallel Computing	CPU	大規模テーブルデータのための並列計算ライブラリ	◎
Modin	Data Analysis	Parallel Computing	CPU	Pandasのワークフローを並列化し高速化するライブラリ	◎
Pandas	Data Analysis	Toolkit	CPU	データ操作と分析のためのPythonライブラリ	◎

3.3.4.1.3.2　深層学習向けソフトウェア

本項では、主な用途が深層学習であるソフトウェアを示す（但し、LLM向けのソフトウェアは3.3.4.1.3.3、Surrogate Model向けのソフトウェアは3.3.4.1.3.4でそれぞれ示す）。PyTorchやTensorFlowなどの深層学習フレームワークに加え、CPUでの推論用途でOpenVINOやONNX Runtimeなどの推論ランタイム・フレームワークも含める。また、CPU性能を最大限発揮するためのライブラリとして、Arm Compute Library, oneDNNなどの深層学習向けライブラリも含める。

ソフト名	用途	レイヤ	デバイス	機能	ARM動作
OpenXLA	Inference	Compiler	CPU	機械学習・深層学習向け計算グラフコンパイラ	○
OpenVINO	Inference	Framework	CPU	汎用推論ランタイム・フレームワーク	◎
ONNX Runtime	Inference	Framework	CPU/GPU	汎用推論ランタイム・フレームワーク	◎
JAX	Inference	Library	CPU	自動微分とJITコンパイル機能を備えた数値計算ライブラリ	○
Triton Inference Server	Inference	Serving	CPU/GPU	汎用AI推論サーバ	◎
diffusers	Inference	Toolkit	CPU/GPU	拡散モデルを利用するためのツールキット	○
PyTorch	Training Inference	Framework	CPU/GPU	深層学習フレームワーク	◎
TensorFlow	Training Inference	Framework	CPU/GPU	深層学習フレームワーク	◎
Arm Compute Library	Training Inference	Library	CPU	Arm CPU向け深層学習用ライブラリ	◎
oneDNN	Training Inference	Library	CPU	深層学習用ライブラリ	◎
Eigen	Training Inference	Library	CPU/GPU	線形代数テンプレートライブラリ	○
DeepSpeed	Training Inference	Parallel Computing	CPU/GPU CPU&GPU	大規模モデルを分散学習・推論するためのライブラリ	△

3.3.4.1.3.3　LLM 向けソフトウェア

本節では、LLMにおけるモデル学習・推論用ソフトウェアを示す。llama.cppやvLLMなどのLLM向け推論ランタイム・フレームワーク、trlやperfといったファインチューニング用ツールを含めている。なお、LLM学習・推論以外の周辺ソフトウェアについては3.3.4.1.3.5に示す。

ソフト名	用途	レイヤ	デバイス	機能	ARM動作
trl	Fine Tuning	Application	CPU/GPU	強化学習によるLLM用ファインチューニングツール	○
peft	Fine Tuning	Application	CPU/GPU	LLMモデル用ファインチューニングツール	○
llama.cpp	Inference	Framework	CPU/GPU CPU&GPU	LLM専用低ビット推論ランタイム・フレームワーク	◎
Kleidi	Inference	Library	CPU	低ビット行列計算ライブラリ	◎
Ollama	Inference	Serving	CPU	GGUFフォーマットのLLM専用推論サーバ	○
Text Embeddings Inference	Inference	Serving	CPU/GPU	Embedding Model専用推論サーバ	△
Text Generation Inference	Inference	Serving	CPU/GPU	LLM専用推論サーバ	△
vLLM	Inference	Serving	CPU/GPU CPU&GPU	LLM専用推論サーバ	◎
transformers	Inference	Toolkit	CPU/GPU	Transformerモデルを利用するためのツールキット	◎

3.3.4.1.3.4　Surrogate Model 向けソフトウェア

本項では、Surrogate Model向けのソフトウェアを示す。DGLやPyTorch GeometricなどSurrogate Modelの実装で良く使われるGNN（Graph Neural Network）開発用フレームワークに加え、PhysicsNeMOのような科学向け深層学習モデル開発ツールを含めることで、AI for Scienceのためのモデル開発を可能とする。

ソフト名	用途	レイヤ	デバイス	機能	ARM動作
DGL	Training Inference	Framework	CPU/GPU	GNN開発用のフレームワーク	◎
PyTorch Geometric	Training Inference	Framework	CPU/GPU	PyTorchで実装されたGNN開発用フレームワーク	◎
PhysicsNeMO	Training Inference	Toolkit	CPU/GPU	科学計算向け深層学習ツールキット	◎

3.3.4.1.3.5　MLOps/LLMOps 向けソフトウェア

本項では、MLOps/LLMOps向けのソフトウェアを示す。複雑化するAIモデルの開発・利用を実現するために、AIモデルの運用を意識したソフトウェアを含める。また、LLMOpsについては学習・推論を除く、LLMを活用するために必要となる周辺ソフトウェアを含めることで、LLMを活用したアプリケーション、AIエージェントの開発・利用を可能と

する。

ソフト名	用途	レイヤ	デバイス	機能	ARM動作
MinIO	MLOps	Application	CPU	S3互換の高性能オブジェクトストレージ	○
MLFlow	MLOps	Application	CPU	機械学習のライフサイクル管理プラットフォーム	○
LangChain	LLMOps	Framework	CPU	LLMを活用したアプリケーション開発フレームワーク	○
LangGraph	LLMOps	Framework	CPU	LLM推論ワークフローを構築・実行するためのフレームワーク	○
LlamaIndex	LLMOps	Framework	CPU	LLMを活用したアプリケーション開発フレームワーク	△
Dify	LLMOps	Application	CPU	LLMアプリを視覚的に構築・運用できるプラットフォーム	○
lm-evaluation-harness	LLMOps	Application	CPU/GPU	LLM精度評価ツール	△
LMCache	LLMOps	Application	CPU/GPU	LLMのKV Cache管理ツール	△
OpenAI Agents SDK	LLMOps	Library	CPU	OpenAI互換API向けPython Agents SDK	△
LiteLLM	LLMOps	Server	CPU	LLM推論用プロキシサーバ	△
DSPy	Prompt	Application	CPU	プロンプト最適化ツール	△
TextGrad	Prompt	Application	CPU	プロンプト最適化ツール	△
Milvus	VectorDB	Application	CPU	ベクトルデータベース	◎
FAISS	VectorDB	Library	CPU	ベクトルデータベース向けライブラリ	◎
OpenHands	AI Agent	Application	CPU	コーディング・ブラウザ操作など自律的に行うAIエージェント	△
AutoGen	AI Agent	Application	CPU	マルチAIエージェント開発用プラットフォーム	△

3.3.4.1.3.6　共通して利用されるライブラリ等

本節では、共通して利用されるライブラリを示す。これらのソフトウェアはアプリケーションやフレームワークなどのバックエンドとして利用される。

ソフト名	用途	レイヤ	デバイス	機能	ARM動作
Numba	Backend	Compiler	CPU/GPU	Python高速化ライブラリ	○
ArmPL	Backend	Library	CPU	Arm CPU向け行列計算ライブラリ	◎

OpenBLAS	Backend	Library	CPU	行列計算ライブラリ	◎
OpenMP	Backend	Library	CPU	マルチスレッドによる並列計算用 ライブラリ	◎
Open MPI	Backend	Library	CPU	マルチプロセスによる並列計算用 ライブラリ	◎
OpenRNG	Backend	Library	CPU	科学技術計算向けの並列乱数生成 ライブラリ	○
oneTBB	Backend	Library	CPU	マルチスレッドによる並列計算用 ライブラリ	◎

3.3.4.1.4　他 WG における AI 関連項目

他のWGにおいても、分野毎にAIの観点で検討が実施されている。主な取組みを以下に示す。各取組みの詳細については、当該WGの基本設計書を併せて参照されたい。

WG	取組み
コデザイン検討会	ハードコデザインに向けたユースケースとして、AIユースケースを含めて検討されている。なお、当該WGで挙がったユースケースのうち、AIソフトの観点で特に重要なものについては本文書の2章にも記載した。
運用システム・利用環境 整備検討会	富岳NEXTの運用システムとして、特に、ジョブキューを前提とするHPC向けの運用システムと、k8s等のコンテナオーケストレータを前提とする運用システムの共存について検討されている。
数値計算ライブラリ・ ミドルウェア検討会	AI向けの演算ライブラリ（Arm Kleidi等）について、SME2対応など、性能観点での実現可能性が検討されている。

参照文献

[1] D. Jonas, F. Federico, B. Jonas, N. Michael, T. Brendan, C. Francesco, E. Timo, H. Roland, A. Abbas, d. l. C. Diego, D. Craig, F. Leslie, G. Cristian, H. Andrea , K. James, “Magnetic control of tokamak plasmas through deep reinforcement learning,” 2022.

[2] “SmartSim,” [オンライン]. Available: https://www.craylabs.org/docs/index.html.

[3] Intel, “インテル® CPU における 高速化された AI の 新時代の到来,” [オンライン]. Available: https://www.isus.jp/wp-content/uploads/pdf/TheParallelUniverse_Issue_55_01.pdf.

[4] W. Junlin, W. Jue, A. Ben, Z. Ce , Z. James, “Mixture-of-Agents Enhances Large Language Model Capabilities,” [オンライン]. Available: https://arxiv.org/abs/2406.04692.

3.3.4.2　加速部

本章では、AI フレームワーク、およびアプリケーション分野を含む NVIDIA のソフトウェアスタックについて記述する。

3.3.4.2.1　AI フレームワーク、ツール、ライブラリ、SDK 等

主要な AI フレームワーク、ツール、ライブラリ、および SDK を以下に示す。

- NVIDIA NGC コンテナレジストリ

GPU に最適化された AI、HPC、データ分析向けのコンテナレジストリ。PyTorch や TensorFlow などの各種フレームワークを含む。
https://catalog.ngc.nvidia.com/

- NVIDIA ディープラーニングライブラリ群
  ディープラーニングの学習および推論ワークロードを最適化するために設計された、GPU アクセラレーション対応ライブラリおよび SDK の集合。
  https://developer.nvidia.com/deep-learning-software
- TensorRT / TensorRT-LLM
  大規模言語モデルを含む学習済みモデルを NVIDIA GPU 向けに最適化する高性能推論ランタイム。
  https://developer.nvidia.com/tensorrt
- NVIDIA NeMo
  LLM、音声、マルチモーダルモデルなどの生成 AI モデルを構築・学習・カスタマイズするためのフレームワーク。
  https://www.nvidia.com/en-us/ai-platforms/nemo/
- NVIDIA NIM (Inference Microservices)
  AI モデルを最適化・スケーラブルな API エンドポイントとしてパッケージ化するマイクロサービス群。
  https://www.nvidia.com/en-us/ai-platforms/nim/
- NVIDIA AI Enterprise
  本番 AI 向けにサポートされたフレームワーク、ツール、インフラストラクチャを提供するエンタープライズ向け AI ソフトウェアプラットフォーム。
  https://www.nvidia.com/ja-jp/data-center/products/ai-enterprise/
- Dynamo
  NVIDIA プラットフォーム上で大規模かつ効率的な AI 推論を実現するために設計されたソフトウェアエンジンおよびオーケストレーションレイヤ。
  https://developer.nvidia.com/blog/inside-the-nvidia-rubin-platform-six-new-chips-one-ai-supercomputer/

3.3.4.2.2 Python ベースのコーディング

Python ベースのソフトウェアコンポーネントを以下に示す。

- cuPyNumeric
  NumPy 互換の分散配列ライブラリ。GPU およびマルチノード環境での数値計算を高速化する。
  https://docs.nvidia.com/cupynumeric/latest/user/usage.html
- CUDA Python
  Python ワークフロー内から CUDA API および GPU アクセラレーションに直接アクセス可能にするツール群。
  https://developer.nvidia.com/cuda/python

3.3.4.2.3 可視化、デジタルツイン

- Omniverse
  リアルタイム 3D および USD ベースのワークフローを用いて、物理的に正確なデジタルツインを構築・シミュレーションするためのプラットフォーム。
  https://www.nvidia.com/ja-jp/omniverse

3.3.4.2.4　科学分野向け AI ソフトウェアスタックの代表例

AI for Science 向けの AI および HPC ソフトウェアスタックの代表例について、以下の分野ごとに後続の各節で示す：

- 気象・気候
- 材料科学
- 生命科学
- 産業／製造／社会／デジタルツイン
- 科学向けオープンモデル

3.3.4.2.4.1　気象・気候

典型的な計算パターンは以下のとおりである。

1. グリッドベースの PDE（有限差分／有限体積／有限要素）
2. ステンシル計算（3D／4D）
3. FFT を用いたスペクトル法
4. 大規模 MPI＋GPU（強スケーリング／弱スケーリング）
5. チェックポイント／リスタート

代表的な AI 手法は以下のとおり整理される。

1. ディープラーニングによる気象予測
   - CNN、Transformer、Graph Neural Network を用いた、大気変数のエンドツーエンド予測
   - 数値予報(NWP)の代理モデル、あるいはそれを補完するものとして活用
2. 時空間モデルによるナウキャスティング
   - 降水および荒天の短期(数分から数時間)予測
   - レーダ/衛星データに基づき、CNN-RNN や Transformer により予測
3. 物理–AI ハイブリッドモデル
   - 物理モデルと機械学習の組み合わせ
   - AI を用いたモデルのバイアス補正、計算コストの高いサブグリッドモデルの置き換え
4. AI によるデータ同化高速化
   - データ同化の各ステップを、機械学習を使って近似
   - ニューラルオペレータや代理モデルを使った、状態推定
5. 超解像・ダウンスケーリング
   - 荒い解像度の気象、気候データの、時空間高解像度化
   - CNN や拡散モデルを用いて実装されることが多い
6. 異常気象・災害予測
   - 台風、豪雨、熱波、暴風などに関する確率的な予測
   - 不確実性定量化と組み合わせて用いられる
7. アンサンブル予報エミュレーション
   - AI ベースのアンサンブル予報の生成や圧縮
   - 計算コストを抑えつつ、高速な確率的予測を可能にする
8. 観測データ処理と品質管理

- 衛星、レーダ、現地観測データの自動解析
- ノイズ除去、異常検知、欠損データの補完、などが含まれる

9. フィジックスインフォームドニューラルネットワーク（PINNs）
- 物理法則による制約を課して学習されたニューラルネットワーク
- 偏微分方程式の近似や、物理的に整合性のとれた予測に用いられる

10. 大気力学のための Neural Operators
- 関数空間間をマッピングするオペレータ
- 大規模な大気流れのモデリングに有用

11. 不確実性定量化
- ベイズニューラルネットワークやアンサンブルベースの AI 手法
- 予測の不確実性とリスク水準の推定

12. 気候パターン認識
- 大規模な気候パターンの AI による同定(テレコネクションなど)
- 長期的な気候解析と傾向検出に応用される

13. 意思決定支援 AI
- 天気予報と AI ベースの最適化、または推薦システムの統合
- 災害対応、エネルギー管理、農業、交通分野で利用される

NVIDIA のソフトウェア例
- NVIDIA HPC SDK（コンパイラ/プロファイリングツール/数値演算ライブラリ/通信ライブラリ等）
- Earth-2 オープンモデル群
  https://www.nvidia.com/en-us/high-performance-computing/earth-2/
  - FourCastNet：風、気温、湿度などのさまざまな気象変数に対して高い予測精度を実現する。従来の主要なアンサンブルモデルを上回り、最先端の拡散モデル手法に匹敵する性能を示しながら、これらの手法より最大 60 倍高速に予測を生成する。
  - CorrDiff：低解像度の大陸規模予測を高解像度の地域スケール気象場へとダウンスケーリングする生成 AI アーキテクチャである。従来手法と比較して最大 500 倍高速に、局地予測に必要な高精細解像度を提供する。
  - Earth-2 Medium Range：Atlas と呼ばれる新しいモデルアーキテクチャにより実現されており、気温、気圧、風、湿度を含む 70 以上の気象変数にわたり、最大 15 日先までの中期予報において高精度な天気予測を可能にする。
  - Earth-2 Nowcasting：StormScope と呼ばれる新しいモデルアーキテクチャにより実現されており、生成 AI を用いて国スケールの予測をキロメートル解像度へと高精細化し、局地的な嵐や危険気象について、0～6 時間先までの予測を数分で生成する。
  - Earth-2 Global Data Assimilation: HealDA と呼ばれる新しいモデルアーキテクチャにより実現されており、気温、風速、湿度、気圧を含む現在の大気状態のスナップショットを世界中の数千地点で生成し、気象予測のための初期条件を提供する。
- Physics NeMo：大規模に物理 AI モデルを構築・学習・ファインチューニングするためのオープンソースの Python フレームワークである。物理法則に基づく因果性とシミュレーション・データおよび観測データを組み合わせた AI サロゲートモデルを構築するためのユーティリティを提供し、リアルタイム予測を可能にする。ま

た、計算流体力学（CFD）、構造力学、電磁気学など、さまざまな物理分野にわたる大規模デジタルツインモデルの構築および検証を支援する。
https://developer.nvidia.com/physicsnemo

3.3.4.2.4.2　材料科学

材料科学における代表的な計算パターンは以下のとおりである。

- 分子動力学（短距離相互作用および長距離相互作用）
- 電子構造計算（疎行列／密行列ソルバ、固有値問題）
- Particle-in-Cell 法、モンテカルロ法

このようなソフトウェアスタックにおいて用いられる代表的な AI 手法は、以下のとおり整理される。

1. 機械学習型原子間ポテンシャル（ML ポテンシャル）
   - DFT／MD データで学習したニューラルネットワークまたはグラフベースモデル
   - DFT に近い精度を維持しつつ、大規模分子動力学シミュレーションを可能にする
2. AI により加速された分子動力学（MD）
   - 機械学習力場を従来型 MD ワークフローに統合
   - より長時間スケールおよび大規模原子系の解析を可能にする
3. 物理ベースシミュレーションのサロゲートモデル化
   - 高コストな FEM、フェーズフィールド、メソスケールシミュレーションを AI モデルで代替
   - 材料設計および最適化における高速評価に利用
4. 電子構造特性予測
   - バンドギャップ、生成エネルギー、状態密度を予測する機械学習モデル
   - 多くはグラフニューラルネットワークやメッセージパッシング型アーキテクチャに基づく
5. ハイスループット材料スクリーニング
   - 大規模組成空間から候補材料を AI で絞り込み
   - 自動化 DFT およびデータベース駆動型ワークフローと組み合わせて利用
6. 逆材料設計
   - VAE、GAN、拡散モデル等の生成モデルにより新規材料を提案
   - 目標特性や性能指標に最適化
7. 構造–物性関係の学習
   - 原子構造と物性との相関をデータ駆動で発見
   - 説明可能性および仮説生成を支援
8. AI によるマルチスケール・マルチフィジックスモデリング
   - 原子スケール、メソスケール、連続体スケールを AI で橋渡し
   - 微細構造進化および巨視的挙動の予測に利用
9. フィジックスインフォームド機械学習（PIML／PINNs）
   - 物理制約および支配方程式を組み込んで学習するニューラルネットワーク
   - 予測のロバスト性および物理的一貫性を向上させる
10. 不確実性定量化およびアクティブラーニング
    - ベイズ機械学習およびアンサンブル手法により予測不確実性を評価
    - 効率的な追加学習データ生成のためのアクティブラーニングループを構築

11. 材料画像および微細構造解析
- 顕微鏡画像およびトモグラフィデータに対する深層学習
- セグメンテーション、相同定、欠陥検出などを実施

12. AI によるプロセス最適化および制御
- 合成、加工、製造パラメータの最適化
- AI と、実験およびシミュレーションワークフローとの統合

NIVDIA のソフトウェア例
- NVIDIA HPC SDK（コンパイラ/プロファイリングツール/数値演算ライブラリ/通信ライブラリ等）
- NVIDIA ALCHEMI：化学および材料科学分野に特化したドメイン特化型 NIM、シミュレーションツールキット（モデル学習・ファインチューニングツールおよび高速化カーネルを含む）、ならびに API 群から構成されるソフトウェア群
  https://developer.nvidia.com/blog/revolutionizing-ai-driven-material-discovery-using-nvidia-alchemi/

3.3.4.2.4.3　ライフサイエンス

ライフサイエンスにおける代表的な計算パターンは以下のとおりである。
- タンパク質／複合体に対する分子動力学（MD）
- 医用画像解析（3D CNN、セグメンテーション）
- オミクス解析（大規模行列演算）
- 生物医学向け LLM／マルチモーダル処理

このようなソフトウェアスタックにおいて用いられる代表的な AI 手法は、以下のとおり整理される。

1. タンパク質構造予測およびモデリング
- アミノ酸配列からタンパク質の 3 次元構造を予測する深層学習モデル
- 単一タンパク質、タンパク質複合体、タンパク質–リガンド相互作用を含む

2. 分子動力学および AI 強化シミュレーション
- 機械学習型力場を用いた AI で加速された分子動力学
- より長時間スケールおよび大規模生体分子系のシミュレーションを可能にする

3. 創薬およびバーチャルスクリーニング
- リガンドベースおよび構造ベースの創薬スクリーニングに用いる AI モデル
- 結合親和性、ドッキングポーズ、薬剤–標的相互作用の予測

4. 薬剤およびタンパク質設計のための生成モデル
- VAE、GAN、拡散モデルを用いて新規分子やタンパク質を生成
- 有効性、選択性、物理化学特性の最適化を実施

5. オミクスデータ解析（ゲノミクス、トランスクリプトミクス、プロテオミクス）
- 高次元生体データへの深層学習適用
- 変異検出、発現解析、経路推定などを実施

6. 医用画像解析
- CNN および Transformer を用いた医用画像（MRI、CT、病理画像）の解析
- セグメンテーション、診断支援、疾患分類への応用

7. マルチモーダルデータ統合
- 分子データ、臨床データ、画像データ、オミクスデータを統合する AI モデル
- 包括的疾患モデリングおよび精密医療を可能にする

8. 大規模言語モデルによる生体配列モデリング
- DNA、RNA、タンパク質配列で学習された LLM
- 機能予測、アノテーション、変異影響解析に利用

9. 細胞および空間生物学解析
- 単一細胞データおよび空間トランスクリプトミクスデータの AI 解析
- 細胞タイプ、状態、組織構造の同定

10. システム生物学およびネットワークモデリング
- 遺伝子制御ネットワークおよびシグナル伝達ネットワークの AI 駆動モデリング
- 複雑な生物学的システムの理解を支援

11. 不確実性定量化およびアクティブラーニング
- ベイズ手法およびアンサンブル手法によるモデル信頼度の定量化
- 実験計画およびデータ取得を導くアクティブラーニング

12. AI 主導の実験設計および自動化
- AI とラボ自動化を組み合わせたクローズドループシステム
- 実験条件およびワークフローの最適化

NIVDIA のソフトウェア例
- NVIDIA HPC SDK（コンパイラ/プロファイリングツール/数値演算ライブラリ/通信ライブラリ等）
- Clara オープンモデル群
  https://github.com/nvidia/clara
  - La-Proteina
    原子レベルで精密な大規模タンパク質の設計を可能にし、研究および創薬候補開発を支援する。従来は治療困難と考えられていた疾患研究に新たな手法を提供する。
  - ReaSyn v2
    製造プロセス設計を創薬プロセスに組み込むことにより、AI 設計薬剤が実際に合成可能であることを保証する。
  - KERMT
    候補薬剤が人体とどのように相互作用するかを予測することで、開発初期段階における高精度な計算機ベースの全性評価を提供する。
  - RNAPro
    RNA 分子の複雑な 3 次元構造を予測し、個別化医療の可能性を拡張する。

- NVIDIA BioNeMo™ Framework ：バイオ医薬分野向け深層学習モデルの構築および学習のためのオープンソース機械学習フレームワーク。ドメイン特化型の学習レシピ、データローダ、事前学習済みかつ最適化済みの AI モデルアーキテクチャ例を含み、高性能を実現するよう最適化されている。これにより、AI モデル構築をより迅速かつ容易にするものである。
  https://github.com/NVIDIA/bionemo-framework

3.3.4.2.4.4　産業、製造、社会、デジタルツイン

産業／製造／社会／デジタルツイン分野における代表的な計算パターンは以下のとおりである。

- CAE（構造／流体／熱／マルチフィジックス）
- サロゲートモデル（CAE 結果のニューラルネットワーク近似）
- 設計空間探索／レコメンデーション
- 工場／都市／設備向けデジタルツイン（3D＋時系列データ）

このようなソフトウェアスタックにおいて用いられる代表的な AI 手法は、以下のとおり整理される。

1. 予知保全（Predictive Maintenance）
   - 設備故障および残存有効寿命（RUL）を予測する機械学習モデル
   - センサデータ、振動データ、音響データ、運転データに基づく
2. 品質検査および欠陥検出
   - CNN および Transformer を用いたコンピュータビジョンによる自動外観検査
   - 表面欠陥、寸法誤差、異常の検出
3. プロセス最適化および制御
   - 製造パラメータ（温度、圧力、速度など）を最適化する AI モデル
   - 強化学習およびサロゲートベース最適化を含む
4. デジタルツインのモデリングおよびシミュレーション
   - AI により高度化された機械、製造ライン、工場のデジタルツイン
   - リアルタイムシミュレーション、モニタリング、What-if 解析
5. 異常検知および故障診断
   - 教師なし学習または半教師あり学習による異常挙動の検出
   - 早期故障検知および根本原因解析に利用
6. サプライチェーンおよび生産計画最適化
   - 需要予測、在庫最適化、スケジューリングに対する AI 適用
   - 不確実性およびリアルタイムデータを考慮
7. ロボティクスおよび自律システム
   - ロボットの知覚、動作計画、マニピュレーションに対する AI
   - 組立、搬送、協働ロボットなどへの応用
8. ジェネレーティブデザインおよびトポロジ最適化
   - AI 支援による最適化された機械設計の生成
   - 性能、重量、製造性、コストのバランスを考慮
9. エネルギー効率およびサステナビリティ最適化
   - エネルギー消費および排出量を最適化する AI モデル
   - 脱炭素および持続可能な製造目標を支援
10. 人間–機械の協調および安全性
    - 作業者の行動および作業環境を監視する AI
    - 安全性向上および Human-in-the-loop 型意思決定を支援
11. 自然言語および知識ベースシステム
    - マニュアル、ログ、運用文書を解析する AI システム
    - トラブルシューティング、教育訓練、意思決定支援に利用

12. クローズドループ学習および継続的改善

- 運用データから AI モデルが継続的に学習するフィードバックループ
- 適応型製造システムおよび継続的最適化を可能にする

NIVDIA のソフトウェア例

- NVIDIA HPC SDK（コンパイラ/プロファイリングツール/数値演算ライブラリ/通信ライブラリ等）
- Physics Nemo
- Omniverse

3.3.4.2.5　科学のためのオープンモデル群

科学向けオープンモデルにおける代表的計算パターンを次の通り整理する

- マルチモーダル科学データ解析（画像＋波形＋テキスト）
- 科学特化型 LLM（論文、コード、実験ログ）
- フィジックスインフォームドニューラルネットワーク／サロゲートモデル
- 不確実性定量化を伴うベイズモデル
- 推論パイプライン最適化

NVIDIA のソフトウェア例

- 推論パイプラインの最適化
  - TensorRT-LLM
    https://developer.nvidia.com/tensorrt
  - Dynamo
    https://developer.nvidia.com/blog/inside-the-nvidia-rubin-platform-six-new-chips-one-ai-supercomputer/
- トレーニング
  - PyTorch / TensorFlow (NGC containers)
    https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch
    https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow
  - CUDA-X DL libraries (cuDNN, cuBLAS, cuTENSOR,等)
    https://developer.nvidia.com/cudnn
    https://developer.nvidia.com/cublas
    https://developer.nvidia.com/cutensor
  - NCCL (前章の通り)
    https://developer.nvidia.com/nccl
- 生成 AI モデル構築
  - NeMo
    https://docs.nvidia.com/nemo-framework/index.html
  - NIM
    https://developer.nvidia.com/nim

3.3.4.2.6　今後の連携

理研と NVIDIA は、AI for Science の推進に向けた協力を確認している。協業の第一段階として、理研のプロジェクトの中から 5 つの重点分野を定義し、NVIDIA が共同で参画するとともに、理研の研究活動を支援することを協議している。

分野 1： 科学向けオープン基盤モデル

分野 2： ドメイン科学：物理

分野 3： ドメイン科学：バイオサイエンス

分野 4： ドメイン科学：材料科学

分野 5： 量子コンピューティング

これらの活動を通じて得られた知見は、適宜、富岳 NEXT AI ソフトウェアワーキンググループの AI ソフトウェアに共同でフィードバックされる。これら二つの取り組みを連携させることにより、最新のワークロードを富岳 NEXT のアーキテクチャおよびソフトウェア設計に反映させるものである。

3.3.4.3　高度化部

本章では、FugakuNEXT システムにおける AI 機能を実現するために必要となる、ソフトウェアおよびハードウェアの中核的要件を整理する。

3.3.4.3.1　CPU・ネットワーク・システムソフトウェアに関する要件

本節では、最新の AI ワークロードを可能とするために必要な、MONAKA CPU、専用インターコネクト、ならびにベンダ提供のシステムソフトウェアに焦点を当てる。

3.3.4.3.1.1　システムソフトウェアおよびミドルウェアのサポート

システムソフトウェアに関する要件は別章で概ね議論されているが、ここでは AI アプリケーションとの接続性について改めて強調する。

Agentic ワークフローや現代的な MLOps をサポートするためには、システムソフトウェアは従来のバッチ処理中心の運用モデルを超えて進化する必要がある。特に **コンテナ化** は不可欠であり、安全性の高い「Build Once, Run Anywhere」を実現するパイプラインが提供されなければならない。

仮にシステムがベアメタル実行を前提とした高性能 HPC 向けフォーマット（例：Apptainer/SIF）を中心に設計される場合であっても、標準的な OCI/Docker イメージを実行可能とするコンテナイメージの変換機能は、AI エコシステムとの親和性を確保する上で必須となる。

オーケストレーション戦略についても近代化が求められる。

標準的な Slurm 上に一時的な Kubernetes 等によるクラスタを展開する「Cloud-over-HPC」方式、あるいは逆にKubernetes 等のクラウド基盤上で Slurm controller を動作させる「HPC-over-Cloud」方式であっても、リソースマネージャは**API 駆動型インターフェース**を提供する必要がある。

これにより、Kubeflow のような MLOps プラットフォームや自律的エージェントといった高次の意思決定システムが、人手による CLI 操作に依存することなく、プログラム的にジョブの投入・監視を行うことが可能となる。

さらに、**データレイヤ**は、しばしば数十億規模の小規模ファイルから構成される AI データセットを円滑に扱える必要がある。従来の POSIX ファイルシステムはこの種のワークロードに不向きであるため、**S3 互換オブジェクトストレージ**をストレージ階層の第一級要素として統合することが求められる。

3.3.4.3.1.2　CPU（MONAKA）およびハードウェアアーキテクチャ

MONAKA CPU が AI 向けの協調プロセッサとして有効に機能するためには、**Scalable Matrix Extension（SME）**および **Scalable Vector Extension（SVE）**が標準的な AI フレームワーク内で完全に統合されている必要がある。

* **フレームワークの同等性および同期性**: PyTorch、TensorFlow、JAX といった主要フレームワークにおいて、NVIDIA GPU スタックと機能的に同等で、かつバージョン整合性の取れた CPU バックエンドが提供されることが必要である。これにより、研究者はデバイスを意識することなく、あるいはハイブリッドなワークフローを、バージョン競合なしに実行できる。
* **ベンチマーク**: Fujitsu は、Transformer アーキテクチャおよび創発性 (emerging) モデルにおいて重要となる FP16 および INT8 の行列演算を中心に、主要計算プリミティブに関する包括的な性能ベンチマークを提供すべきである。

本アーキテクチャは、**統一メモリ（Unified Memory）**モデルをサポートし、「メモリ拡張型 AI（Memory-Extended AI）」を実現する必要がある。これにより、GPU から CPU 主記憶への高帯域・ゼロコピーアクセスが可能となり、巨大な埋め込みテーブルや大規模グラフ解析など、GPU の HBM 容量を超えるワークロードの処理が可能となる。

さらに、ランタイムは「ヘテロジニアス分割計算（Heterogeneous Split Compute）」をサポートし、複雑な分岐処理を CPU が、密な数値計算を GPU が担当しつつ、従来の PCIe 転送に伴う性能ペナルティを回避した CPU-GPU 間の低遅延同期を実現する必要がある。

推論用途において、MONAKA は以下の二つの異なる実行パターンをサポートすべきである。

1. **容量主導型推論**: 大規模言語モデル（LLM）の膨大な重みをシステムメモリにオフロードし、必要に応じて GPU にレイヤをストリーミングする方式。
2. **インライン／サロゲートモデル推論**: 推論に特化して最適化された軽量フレームワーク（例：ONNX Runtime、専用 C++ ライブラリ）を用い、事前学習済みのサロゲートモデルを、実行中の数値シミュレーション（C++/Fortran ホストコード）内部で継続的に実行する方式。この「インライン推論」は、メインのシミュレーションループを停滞させない最小限のオーバーヘッドで実現されなければならない。

3.3.4.3.1.3　ネットワークおよびインターコネクト

分散学習の効率を確保するため、FugakuNEXT の専用ファブリックに最適化された**通信ライブラリ**が必要である。これらのライブラリは NCCL と同等の API を提供し、マルチノード学習および推論に不可欠な低遅延コレクティブ通信をサポートすべきである。

データ取り込み（Data Ingestion）は、現代的 AI が扱う大規模データセットに対応するための、堅牢かつ専用のインフラを必要とする。設計には、高性能 CPU と大容量メモリバッファを備えた **データ取り込みノード**を含めるべきである。これらのノードは、内部クラスタファブリックとは独立した**外部ネットワーク（WAN）インターフェース**を備える必要がある。

ソフトウェア面では、**Globus** や **Apache Kafka** などの高性能・並列データ転送ツールをサポートし、外部データソースからクラスタ内オブジェクトストレージへの大規模データ移動を自動化・最適化できることが求められる。

3.3.4.3.1.4　ベンダーサポートおよび統合

円滑な移植を実現するためには、ベンダとの緊密な協調が不可欠である。早期のモデル適応を可能とするため、

ハードウェア試作機、エミュレータ、あるいは性能シミュレータへの **早期アクセス**が提供されることを要求する。また、現行の実行トレースに基づき Blackwell 世代 GPU 相当の性能を推定するなど、性能予測に関する協調的検討も求められる。

コンパイラツールチェイン（コンパイラ、プロファイラ、デバッガ）は **形式手法（Formal Methods）**と統合されるべきである。特に、**TADASHI** のようなシステムが正当性の保証されたコード変換を生成できるよう、ポリヘドラルモデル抽出をサポートする必要がある。加えて、JAX や Numba を介した JIT コンパイラのサポートにより、科学 AI アプリケーションにおける動的カーネルへの対応が求められる。

3.3.4.3.1.5　ワーキンググループ間の協調

本システムソフトウェアの発展には、FugakuNEXT の他のワーキンググループとの緊密な連携が不可欠である。とりわけ、プラットフォームにおけるオーケストレーション戦略およびコンテナ化戦略は、HPC 向けキューとコンテナオーケストレータの共存を担う運用システム・利用環境整備 WG との整合が求められる。また、AI の利用要件は Codesign Review WG を通じてハードウェアコデザインに直接的に反映されるとともに、数値計算ライブラリ・ミドルウェア WG が Arm Kleidi をはじめとする AI 指向計算ライブラリの性能面での実現可能性を評価する。

3.3.4.3.1.6　CPU が優位性を持つ推論およびワークロード

GPU が大規模学習の主要な担い手となる一方で、MONAKA CPU は、数理的あるいは運用上の観点から GPU を上回る性能を発揮しうるワークロードにおいて、積極的に活用されなければならない。具体的には、GPU のオーバーヘッドに対して計算量が相対的に小さい処理（小バッチ推論や小規模言語モデル（SLM）など）、ならびに頻繁な条件分岐を伴うランダムフォレストのような GPU アーキテクチャに不向きなワークロードに対して、アーキテクチャの最適化が必要である。加えて、強化学習における報酬計算（例：SmartSim 経由）の効率的な実行や、RAG におけるエンベディングモデルや投機的デコーディングにおけるドラフトモデルといった、ハイブリッド構成における小規模な補助モデルの管理についても CPU が担うべきである。

3.3.4.3.1.7　OSS エコシステムおよび Arm 互換性

上述の CPU 指向 AI ワークロードを実現するため、システムは Arm 向けに最適化された堅牢かつ事前コンパイル済みのオープンソースソフトウェア（OSS）スタックを提供しなければならない。具体的には、XGBoost、Scikit-Learn、NumPy、Pandas、Modin といった基盤的な機械学習・データ分析フレームワークに対して、検証済みのネイティブサポートが必要である。さらに、OpenVINO、llama.cpp およびベクトルデータベース（Milvus、FAISS）などの深層学習・LLM 関連コンポーネント、ならびに DGL、PyTorch Geometric、PhysicsNeMO 等のサロゲートモデルツールキットについても、ネイティブ環境でのサポートが求められる。

3.3.4.3.2　GPU に関する要件

本節では、GPU アクセラレーション層、関連ライブラリ、ならびにエコシステム全体との統合に焦点を当て、世界最先端 AI スーパーコンピュータの中で FugakuNext がその競争力を維持するための要件を整理する。

3.3.4.3.2.1　GPU アクセラレーションおよびライブラリ

プラットフォームは、cuBLAS、cuDNN、CUTLASS などのライブラリを通じて、主要計算プリミティブに対する標準的かつ最適化されたサポートを提供し、PyTorch や TensorFlow といった主要学習フレームワークとの完全な互換性を確保する必要がある。

3.3.4.3.2.2　推論ソフトウェアおよびエンジン

NVIDIA は、**vLLM** や **TensorRT-LLM** といった高性能推論エンジンに対する堅牢なサポートを保証しなければならない。このソフトウェア層は極めて重要であり、以下のような現代的最適化機能を標準でサポートする必要がある。

- 最大スループットを実現する **連続バッチング**
- 効率的なメモリ管理を可能とする **ページド アテンション(Paged Attention)**
- マルチターンエージェントや RAG を高速化する **プロンプトキャッシュ(Prompt caching)**（例：Radix Tree）
- 生成遅延を低減する **投機的デコーディング(Speculative Decoding)**

これらの推論エンジンは、FugakuNEXT 固有のハードウェア構成に最適化されていなければならない。
さらに、AI-for-Science を支援するため、ベンダは**軽量インライン推論フレームワーク**を提供すべきである。

- **サロゲートモデル実行**: 高性能シミュレーションコード（C++/Fortran）に直接組み込まれ、PINN やエミュレータ等のサロゲートモデルを極低遅延で実行可能なライブラリ。
- **ゼロオーバーヘッド統合**: フレームワークは、コンテキストスイッチのオーバーヘッドを最小化し、シミュレーション状態と推論エンジン間でゼロコピーのデータ転送を可能とすること。

標準ライブラリに加え、創発性(Emergent)アーキテクチャへの追従も重要である。例えば、Mamba に代表される状態空間モデル(SSM)やエネルギーベースモデルへの移行が進む場合、ベンダはこれらの科学的ニーズに対応するため、迅速にカーネル更新およびベンチマークを提供すべきである。また、移植戦略を検討するため、CUDA のような独自技術と OpenACC 等のオープン標準との性能比較に関する透明性の高いベンチマークも要求される。

3.3.4.3.2.3　MONAKA との統合

システムの結合性を確保するため、ベンダは**ハイブリッド一貫性（Hybrid Consistency）**を実現する必要がある。これは、Grace-Hopper プラットフォームに類似した CPU-GPU 統合を MONAKA 環境向けに実装し、Unified Memory や GPU Direct Storage といった機能を適応させることで、NVIDIA のツールチェーンが Fujitsu のハードウェアと簡潔かつ効果的に相互運用できる状態を指す。

3.3.4.3.2.4　AI for Science の実現

「AI-for-Science」への要件を満たすため、ソフトウェアスタックは、主要な科学領域が持つ固有の計算パターンや方法論に適合するよう設計されなければならない。

気象・気候： 深層学習に基づく気象予報、時空間ナウキャスティング、および超解像ダウンスケーリングへの対応が求められる。高解像度かつ予測的な気象モデリングの実現に向けて、Earth-2（FourCastNet、CorrDiff、StormScope を含む）等のオープンモデルとの統合が必要である。

材料科学： 原子間ポテンシャルを活用した分子動力学計算および電子構造計算の、機械学習による高速化が求められる。また、生成モデルを用いた逆材料設計や高スループットスクリーニングに対応するワークフローの整備が必要である。

生命科学： タンパク質構造予測、オミクスデータ解析、および生成的手法による創薬設計に最適化されたインフラの提供が求められる。BioNeMo や Clara といったフレームワークを統合し、オープンモデル（例：La-

Proteina、RNAPro）を使用することにより、大規模な生体分子シミュレーションおよびマルチモーダルデータの統合的解析を可能とすべきである。

産業・デジタルツイン：Omniverse や PhysicsNeMO 等のツールを活用し、複雑な物理現象のモデリング、予測保全、生成設計、およびリアルタイムなデジタルツインシミュレーションの実行を可能とする必要がある。

3.3.4.3.2.5　協調研究および将来ロードマップ

ソフトウェアアーキテクチャが研究者の最先端の要求に継続的に応え続けることを確保するため、RIKEN と NVIDIA は詳細設計段階から協調研究イニシアティブを開始した。本パートナーシップでは、科学、物理学、生命科学、材料科学、量子コンピューティングの 5 つの重点領域におけるオープンファウンデーションモデルに関するパイロットプロジェクトを共同で推進する。各領域の実証的なアプリケーションを通じて得られる知見と最適化の方策は、FugakuNEXT AI ソフトウェアワーキンググループへ継続的にフィードバックされ、ハードウェアアーキテクチャの設計指針に反映されるとともに、ソフトウェアスタックのさらなる高度化に活用される。

3.3.4.3.3　RIKEN に関する要件（上位ソフトウェアスタックおよび利用形態）

本節は高度化レイヤを構成する要件であり、Fujitsu/NVIDIA が提供する基盤の**上位**に位置する AI アプリケーションスタック、マネージドサービス、ならびに Agentic ワークフローに焦点を当てる。

3.3.4.3.3.1　AI サービス（推論および学習）

推論は **Managed Inference as a Service** として提供されるべきである。ベンダが提供する最適化済み推論エンジンを基盤とし、RIKEN は API（HTTP/gRPC）を通じて、バージョン管理されたモデルカタログへアクセス可能なサービス層を構築する。サービスバックエンドは、負荷やマルチテナントアクセス管理に応じた**オートスケーリング**を担う。特に重要なのは、研究者が独自モデルを安全なサンドボックス内でアップロード・提供できる **BYOM（Bring Your Own Model）**のサポートである。

同様に、**ファインチューニング**についても、インフラの複雑性を抽象化するマネージドサービスとして提供されるべきである。ユーザは UI/API を通じて分散学習やチェックポイントを設定でき、PEFT（LoRA/QLoRA）およびフルファインチューニングの双方をサポートする必要がある。これを支える基盤として、NVIDIA GPU および MONAKA SME の双方を対象とした量子化・プルーニング・コンパイルを行う**モデル最適化ツールチェーン**が求められる。

3.3.4.3.3.2　AI 支援開発および Agentic ワークフロー

AIによる自律的開発を支援するため、特定の **Agentic インフラ**が必要となる。これは、「レベル 3 抽象化」あるいは「制御プレーン」実行モデルとして、エージェントが生のシェルコマンドではなく、submit などの事前定義されたラッパーを通じて操作する仕組みを指す。

システムは、エージェント向けに**モックターゲット**（軽量 CPU 専用環境）および**エフェメラル(一時的)サンドボックス**（高速コンテナ実行環境）を提供し、1 分未満でのコンパイル・テストサイクルを可能とすべきである。

コード生成における**ハイブリッド推論戦略**として、高度な計画立案には最先端モデルへの安全なゲートウェイを用い、機密性の高い内部コード生成には自己ホストされたファインチューニング済みモデルを用いる。

最後に、コード品質を保証するため**検証フライホイール**を実装する。これには、Benchpark 等を用いた**継続的性能検証**と、得られる実測指標に基づく最適化の検証、および TADASHI（ポリヘドラルモデル）を用いた**正当性保証**による検証ループの形態変換が含まれる。ユーザによるトレーニングは、これら構造的な判断を適切に誘導し、AI 生成コードをレビューする能力の向上に重点を置き、人間が重要な設計決定に関与し続けることを保証す

る。

3.3.4.3.3.3　移植性向上のためのパイロットプロジェクト

ベンダには各々の提供するハードウェアに最適化されたカーネルの提供を期待する一方で、CUDA のみに依存する強いベンダーロックインを回避し、AMD GPU など多様なハードウェア上で動作可能なクロスプラットフォームソフトウェアスタックの採用を目指すべきである。

この観点から、Mojo は有力な候補と考えられる。しかし、コードベース全体を即座に移行することは大きなリスクを伴う。そのため、まずは Mojo の有効性を評価する パイロットプロジェクトを立ち上げることを提案する。本取り組みの一環として、技術的実現性と性能向上の可能性とのバランスを考慮しつつ、移植対象とするソフトウェアの限定的なサブセットを慎重に選定する必要がある。

3.3.4.3.3.4　協調研究および将来ロードマップ

ソフトウェアアーキテクチャが、研究者の最先端の要求に継続的に応え続けることを保証するため、RIKEN と NVIDIA は協調研究イニシアティブの詳細設計段階を開始した。本パートナーシップでは、科学、物理学、生命科学、材料科学、量子コンピューティングの 5 つの重点領域におけるオープンファウンデーションモデルに関するパイロットプロジェクトを共同で推進する。各領域の実証的なアプリケーションを通じて得られる知見と最適化の方策は、FugakuNEXT AI ソフトウェアワーキンググループへ継続的にフィードバックされ、ハードウェアアーキテクチャの設計指針に反映されるとともに、ソフトウェアスタックのさらなる高度化に活用される。

3.4 ストレージとデータ管理

3.4.1 はじめに

本章では、本資料の目的、基本設計概要および方針、ストレージ要件について述べる。

3.4.1.1 本資料の目的

本資料は、次世代計算基盤「富岳NEXT」基本設計のストレージ検討会による検討内容の報告書である。本報告書では、2030年頃を見据えた外部記憶装置の技術・性能動向に基づき、システム全体のストレージアーキテクチャの基本設計案、設計目標値、およびその妥当性と根拠について記述する。

3.4.1.2 基本設計概要および方針

3.4.1.2.1 概要

「富岳NEXT」のストレージ基本設計は、科学技術研究分野、産業分野、および社会からの利用ニーズに応えるべく、2030年代の計算環境に最適なストレージシステムの選択肢を探索し、その実現可能性と妥当性を提示することを目的とする。システム全体やその構成要素としてのストレージアーキテクチャについて、現在の技術水準から将来展望を見越した網羅的な調査検討を実施した。

次世代計算基盤では、超並列I/O性能、データ集約型計算への対応、および長期的な運用安定性を確保するストレージシステムが求められる。「富岳NEXT」のストレージシステムは、計算ノード直結の高速なローカルストレージ（第1階層）と、全計算ノードで共有される大容量ストレージ（第2階層）からなる階層化構成を基本とし、各階層で要求される性能要件（帯域幅、IOPS、容量）の実現を目指す。基本設計では、ハードウェアおよびファイルシステムのアーキテクチャ、ストレージシステム活用支援技術、および性能評価手法に関する詳細かつ多角的な検討を行った。

3.4.1.2.2 方針

「富岳NEXT」のストレージ基本設計に係る調査検討は、実現可能性調査に留まらず、主要ストレージベンダーから将来技術のロードマップ情報を包括的に入手し、2030年頃の技術を念頭に置いて選択可能候補を網羅的に比較・検討することを基本方針とする。基本設計の検討は以下の項目に沿って実施され、詳細は該当章に記載する。

- ストレージアーキテクチャの検討（3.4.2 章）：「富岳 NEXT」の利用シーンを明らかにした上で、第 1 階層・第 2 階層の要求仕様に適合するシステム構成、運用管理上期待される機能と性能要件、実現可能性を検討する。
- ストレージシステム活用支援技術の検討（3.4.3 章）：第 1 階層と第 2 階層間のデータ転送技術、外部ストレージ連携手法、およびストレージシステムの運用管理・性能分析ツールについて検討する。
- 性能測定方針とベンチマーク手法の検討（3.4.4 章）：第 1 階層、第 2 階層の性能測定方針およびファイルシステムのベンチマーク手法を検討する。
- コスト推定 （3.4.5 章）：複数予算や段階導入への対応に向けて、ハードウェアを中心に技術要素ごとの選択肢を検討する。
- 詳細設計への申し送り（3.4.6 章）：基本設計の検討結果を踏まえ、詳細設計において更なる精緻化や判断が求められる事項を整理する。

なお、主要ストレージベンダーからのヒアリングに当たり、理化学研究所、富士通との間で三者間NDAを締結した。ヒアリング結果を容易に比較するため、本検討会での議論内容や、「次世代計算基盤に係る調査研究事業　委託業務成果報告書」（以降FSと呼ぶ）の課題をヒアリングシートの形式にまとめ、各ベンダに送付して回答を収集した。ヒアリングシートを使うことで、各々のストレージベンダーの提供情報の粒度を統一したことが成果である。

3.4.1.3　富岳 NEXT ストレージ要件

本章では、富岳NEXT基本設計仕様書の要件と、本報告書の記載箇所のマッピングを表 3-24に記載する。

表 3-24　仕様書要件と本報告書のマッピング

仕様書記載項目			仕様書記載内容	本報告書の章番号
一般事項	3.納入物	(1)成果報告書	全体システム構成（全体構成図、計算ノードの設計仕様案。特に外部記憶装置に関しては 2030 年頃の技術・性能を踏まえた検討結果とその妥当性・根拠等。）	3.4.2.1 3.4.2.3.1 3.4.2.4.1 3.4.5.2
	18.用語	(1)単位	演算処理性能、ストレージ容量及び通信バンド幅は 10 のべき乗値、主記憶容量は 2 のべき乗値とする。	―
技術仕様	II-1 1. 開発要件	5)ストレージの要件	計算ノード又はジョブローカルで使用できる高速ストレージ又はキャッシュと、全計算ノードで共有するストレージシステムを持ち、各階層において要求される帯域、IOPS、容量を実現するための構成にすべきである。また、実運用で長時間稼働させたとしても安定した性能が維持されることとする。	3.4.2.1 3.4.2.3.1 3.4.2.4.1 3.4.4.1 3.4.4.2 3.4.5.2
	II-1 2. 全体システム構成の仕様	(1)「富岳 NEXT」の構成	主にアプリケーションソフトウェアを処理する機能を提供する本体システム、主にプログラム開発作業などユーザ作業を処理する機能を提供するフロントエンドシステム、及びファイルシステムを含む周辺装置で構成されるものとする。	3.4.2.1
		(2)本体システムの全体仕様 5)	各計算ノードはファイルシステムにアクセスできること。	3.4.2.1
		(3)計算ノードの仕様 10)	計算ノードからアクセス可能なファイルシステムおよび階層化ファイルシステムを想定すること。	3.4.2.1
	II-1 2. (5) 周辺装置		周辺装置として、外部記憶装置及び外部ネットワーク接続用機器等全体システム動作に必要な機能を想定すること。	3.4.2.1
		1)	外部記憶装置は、アクセス速度、容量等を考慮したシステム構成を持つものとする。アクセス速度に関してはファイルシステムを介した実効性能とする。	3.4.2.3.1 3.4.2.3.3 3.4.2.4.1 3.4.2.4.4 3.4.5.2
			①SSD の使用を基本とした 2 階層の階層化ストレ	3.4.2.1

			ージシステムを導入し、段階的に拡張ができること。	3.4.2.4.2
			②第 1 階層はローカルストレージを導入し、以下の性能を目標として基本設計で決定すること。 ・バンド幅：計算ノードの全メモリの内容を 2 分以下で書き出すことができること。読み込みも同等以上の性能であること。 ・IOPS：単一計算ノードあたりメタデータ処理（ファイル生成・削除、Read/Write を含む）IOPS が 8K 以上であること ・容量：総メモリサイズの 3 倍以上となること ・ローカルストレージについて、各計算ノードのローカルに搭載するノードローカルストレージ、及びストレージ専用のノードを導入し複数台の計算ノードからネットワークを介して共有されるニアノードローカルストレージなどを第 1 階層のストレージとして想定すること。	3.4.2.3.1 3.4.2.3.3 3.4.5.2
			③第 2 階層は全ての計算ノードから共有される共有ストレージを導入し、以下の性能を目標として基本設計で決定すること。 ・バンド幅：計算ノードの全メモリの内容を MPI-IO による Single Shared File 方式によって 10 分以下で書き出すことができること。読み込みも同等以上の性能であること。 ・想定する Single Shared File 方式： 各プロセスが同じファイルの別々の領域にアクセスするパターン。ファイルは 2 次元以上の構造化データ (2D データ等)を想定すること。 ・IOPS：全ノードから同時にリクエストされる I/O 処理に対して、I/O 処理が停止することなく安定して稼働し 1 ノードあたり IOPS が 1K 以上の性能であること。 ・容量：総メモリサイズの 30 倍以上となること バンド幅、IOPS、容量、価格のバランスを最適化するために、SSD を基本としながらも HDD を併用した第 2 階層ストレージを想定すること。	3.4.2.4.1 3.4.2.4.4 3.4.5.2
		3)	運用において、長時間にわたって安定した性能を維持するための設計とその安定性を定量的に評価するためのベンチマーク手法を検討し、検討結果を成果物に記載すること。	3.4.4.1 3.4.4.2
		4)	ファイルシステムについては、ユーザのフィードバックに対して継続的に改修ができるよう OSS を基本とすること。	3.4.2.3.4 3.4.2.4.5
	II-2 1. 2 外部記憶装置	1)	階層数と階層の実現方式	3.4.2.1 3.4.2.3.1 3.4.2.4.1 3.4.3.1
	II-2 1. 2 外部記憶	2)	想定される複数の予算額において各階層において以下を検討し、検討結果を成果物に記載すること	3.4.5.2

	装置		a. 使用するデバイスの種類 b. 容量、バンド幅、IOPS c. 冗長化方式 d. 拡張性・段階導入の可否 e. 消費電力 f. UPS の必要性 g. ラック（規格、ラック数、ラックサイズ）やストレージサーバー数 h. 予算の内訳	
	II-2 1. 3 システムソフトウェア	6)	アプリケーション全体、アクセラレータ、ストレージシステム、通信ライブラリなど個々の機能についてのプロファイル手法を検討し、必要な開発項目を整理して、成果物に記載すること。	3.4.3.3
	II-2 1.4 設置条件	(2)	周辺装置の機器構成、ラック数、電力などを検討し、検討結果を成果物に記載すること。	3.4.5.2
		(3)	計算ノードおよび周辺機器への電源構成(3 相 4 線式 3P+N+E/415V 又は 480V)を検討し、検討結果を成果物に記載すること。	3.4.5.2
	II-4 システム諸元		以下の項目について検討し、検討結果を成果物に記載すること。 (1) 設置形状、設置面積、消費電力、冷却条件等	3.4.5.2

また、令和4年度から6年度に実施したFSの中から、ストレージ・ファイルシステムに関する課題を抽出し、本基本設計では管理番号（STK-xxx）を付与している。FS課題と本報告書の記載箇所のマッピングを表 3-25に記載する。基本設計では各課題に対してベンダーヒアリングを行い、情報収集を実施した。具体的な対応方法の検討については詳細設計で実施する。

表 3-25 FS 課題と本報告書のマッピング

管理番号	課題内容	課題の論拠	本報告書の章番号
STK-001	SSF性能低下（高並列ジョブで遅延増大、ロック獲得オーバーヘッド）	令和5年度ユーザーヒアリング（asuca, Athena++, R2D2）	3.4.2.3.3 3.4.2.4.4
STK-002	メタデータ性能低下・IOエラー（同一ディレクトリ大量アクセスで性能劣化、evict発生）	令和5年度ユーザーヒアリング（FFB/FFX, R2D2）	3.4.2.3.3 3.4.2.4.4
STK-003	ファイルopen/close遅延（多数プロセス同時アクセスで一部応答が遅延）	令和5年度ユーザーヒアリング（UTHeart、原因未調査）	3.4.2.3.2 3.4.2.4.3
STK-004	大量ファイル作成でスケール不全（MDS性能不足で2000ノード規模に制限）	令和5年度ユーザーヒアリング（Genomon）	3.4.2.3.3 3.4.2.4.4
STK-005	システム安定性不足（MDS負荷集中でIOエラーが発生、ジョブ失敗・再実行）	令和5年度ユーザーヒアリング（複数アプリ）	3.4.2.3.3 3.4.2.4.4
STK-006	MPI-IO非同期I/O未サポート（計算とI/Oのオーバーラップ不可）	令和5年度ユーザーヒアリング（kinaco）	3.4.2.3.3 3.4.2.4.4
STK-007	ストレージ容量不足・クオータ制限（出力抑制、成果創出を妨げる）	令和5年度ユーザーヒアリング（R2D2, kinaco）	3.4.2.3.2 3.4.2.4.3
STK-008	I/O性能限界によりユーザがアプリ改変を強いられる（計算結果が無駄になるリスク）	令和6年度ユーザーヒアリング（富岳ユーザ全般）	3.4.2.3.1 3.4.2.4.1
STK-009	llio_transferの制約（ノード近傍SIOへの配置不可、独自ステージングが必要）	令和5年度ヒアリング（NICAM）	3.4.3.1
STK-010	外部ストレージ転送負荷（HPCI共用や外部研究ストレージへの移動頻発、速度不足）	令和5年度ユーザーヒアリング（R2D2, kinaco）	3.4.3.2
STK-011	ストレージシステム開発が終了し、ユーザーフィードバックに基づく改修ができない	令和6年度報告書記載	3.4.2.3.4 3.4.2.4.5
STK-012	持続的なソフトウェア開発体制不足（運用後も改修・改善が必要）	令和6年度報告書記載	3.4.2.3.4 3.4.2.4.5
STK-013	ハードウェア仕様にストレージ要件が反映されず、I/OノードSSD搭載不足で性能限界	令和6年度報告書記載	3.4.2.3.1

3.4.2 アーキテクチャ

本章では、ストレージシステムの全体構成と利用シーン、第1階層/第2階層それぞれのシステム構成や期待される機能、要求性能、ファイルシステムの候補選定について述べる。

3.4.2.1 全体構成

「富岳NEXT」のストレージシステム全体像を図 3-25に示す。2階層の階層化ストレージシステム構成とし、各計算ノード（アプリケーション）は第1階層/第2階層いずれのストレージシステムにもファイルシステムを介してアクセス可能とする。図中で、第1階層＃1から＃3は各アプリケーションの用途に応じてストレージシステム側の機能を使い分けることを意味する。単一の第1階層ストレージシステムが第1階層＃1から＃3の役割を担う可能性もある。

第1階層/第2階層の間はステージングやキャッシング機能によってデータを転送する。クラウドストレージなどの外部ストレージへはデータ同期機能により転送する。階層間のデータ転送については3.4.3章に記述する。

第1階層/第2階層で使用するデバイスなどのハードウェア構成、およびファイルシステムの検討については2章に記述する。

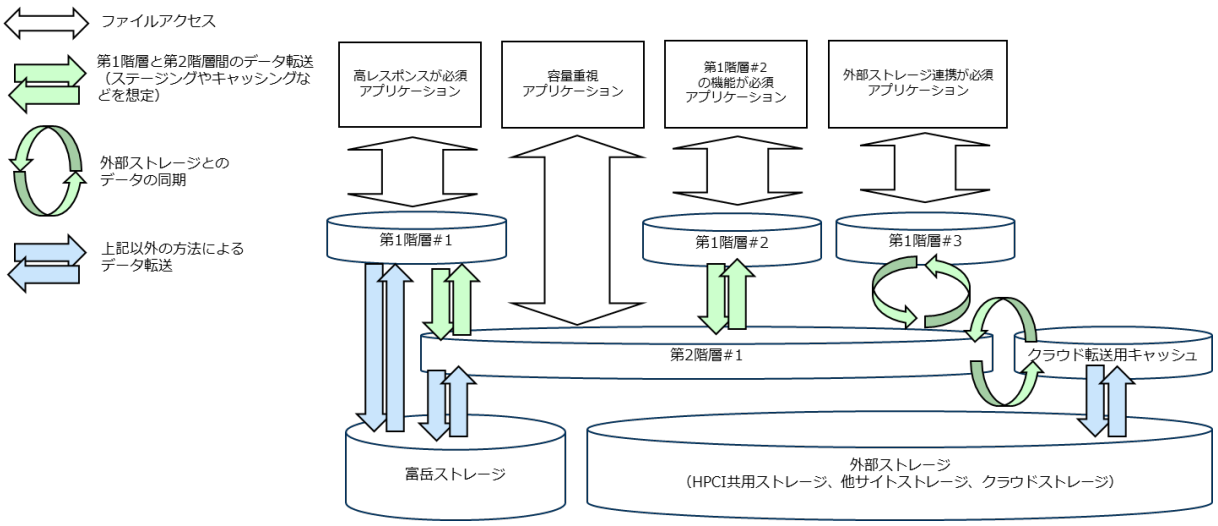


図 3-25 全体構成

3.4.2.2 利用シーン

「富岳NEXT」のユーザが、階層のストレージシステムを利用する方法として想定される現段階の主な利用シーンを図 3-26から図 3-33に示す。なお、図中の色付きの部分はデータ転送が発生することを意味し、矢印の種類及び色の意味については図 3-25と同様である。

- 利用シーン 1
  - 「富岳」からアプリケーションを移行するケース、および、「富岳 NEXT」運用開始前の試行運用のケースを想定する。「富岳」からの移行では動作確認を目的とし、1 ノード規模での演算実行を想定する。「富岳 NEXT」運用開始前の試行運用は、「富岳」の共用開始前にも重要アプリケーションの動作確認やテストベッド環境が要望された経緯があり、「富岳 NEXT」の運用開始前にも同様の要望があることを想定する。試行運用は共用運用開始前のため、利用ユーザは開発者などを想定する。

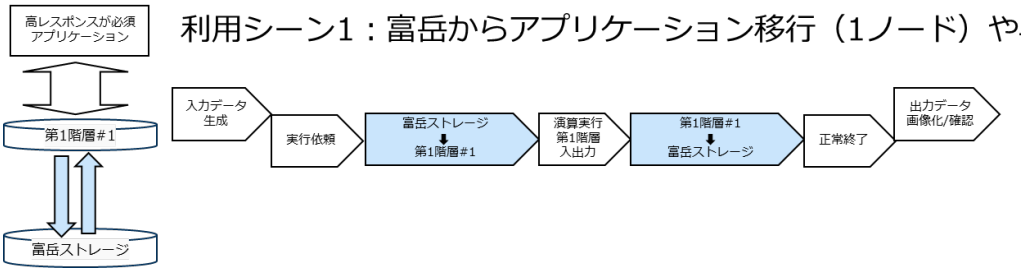


図 3-26 利用シーン 1

- 利用シーン 2
  - 入出力データの多いアプリケーションで入出力を行うファイルシステムを第 2 階層のみに固定して利用するケースや多量ファイルを扱う「富岳」のアプリケーションの移行を想定する。アプリケーションのファイルシステムへの要求性能が低く、第 1 階層を利用してもアプリ性能に寄与しないワークロードにおいて利用される。多量ファイルを扱うワークロードでの利用も想定されるため、第 2 階層の負荷を上げる要因となる可能性がある。

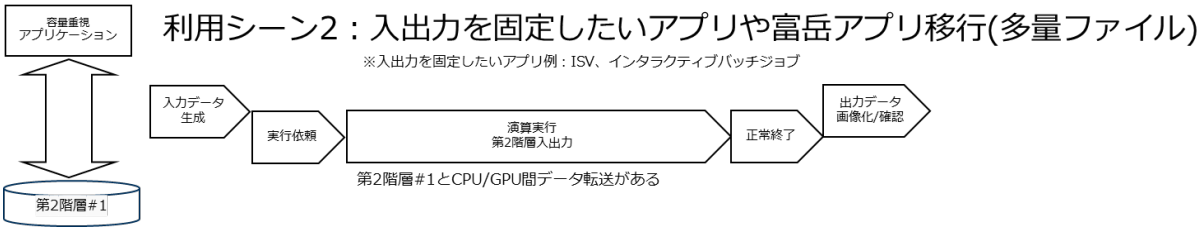


図 3-27 利用シーン 2

- 利用シーン 3
  - 第 1 階層で異なるファイルシステムの独自機能を利用するケースを想定する。例えば、通常は第 1 階層#1 のファイルシステムを使用するが、あるアプリケーションでは第 1 階層#2 のファイルシステムを使用するケースである。アプリケーションの実行前に利用するファイルシステムを切り替える。その後、アプリケーションの演算実行を行い、出力結果を第 2 階層にデータ転送する。最後にファイルシステムを元の状態に戻す。

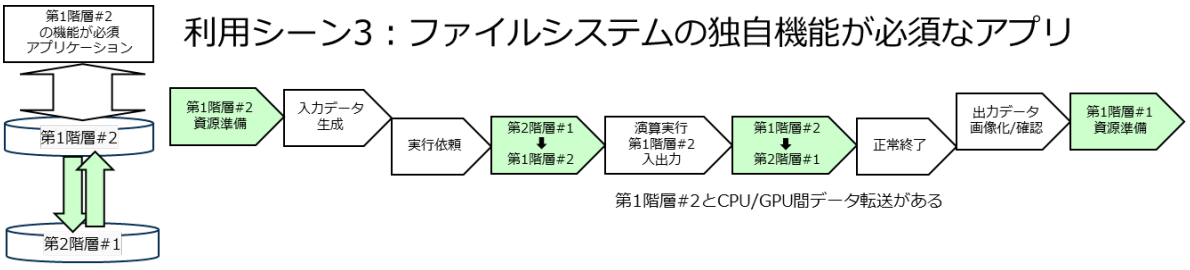


図 3-28 利用シーン 3

- 利用シーン 4
  - 高レスポンスが要求されるアプリケーションなどで、第 1 階層を利用して演算実行するケースを想定する。ステージングやキャッシュなどによって演算実行前後に第 2 階層から第 1 階層にデータの転送を行う。アプリケーションのファイルシステムへの要求性能が高く、第 1 階層を利用することでアプリ性能が得られるワークロードにおいて利用されると考えられる。

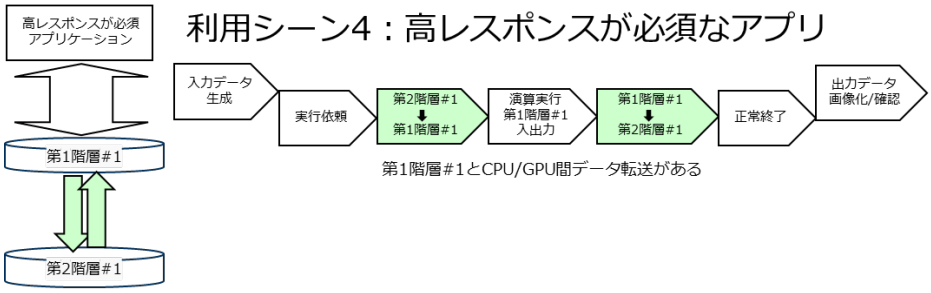


図 3-29 利用シーン 4

- 利用シーン 5
  - アプリケーションのファイルシステムへの要求性能が高く、入力データが多量にあり、高頻度かつ連続的に演算結果を出力することで、第 1 階層の容量が不足するようなケースを想定する。このケースでは高頻度かつ連続的に出力される演算結果をワークロードの間で定期的に第 2 階層にデータ転送が必要となる。
    第 2 階層から第 1 階層への入力データのデータ転送は 1 回のみ行い、その後の出力データの第 1 階層から第 2 階層へのデータ転送は複数回行うことを想定している。

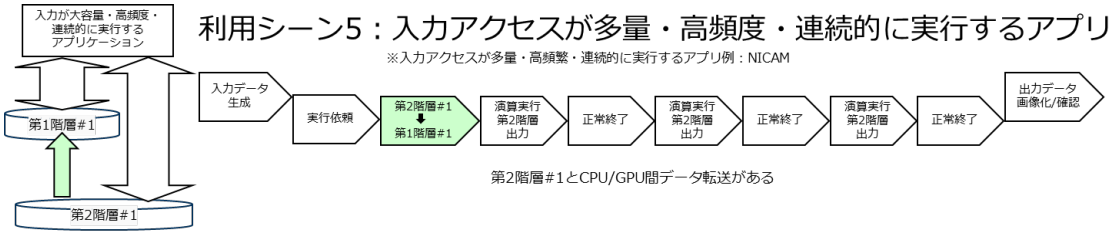


図 3-30 利用シーン 5

- 利用シーン 6
  - 複数ノードを利用するアプリケーションの移行時に、「富岳」のストレージから第 2 階層を経由してデータ転送を行うケースを想定する。「富岳」を利用しているユーザを想定する。

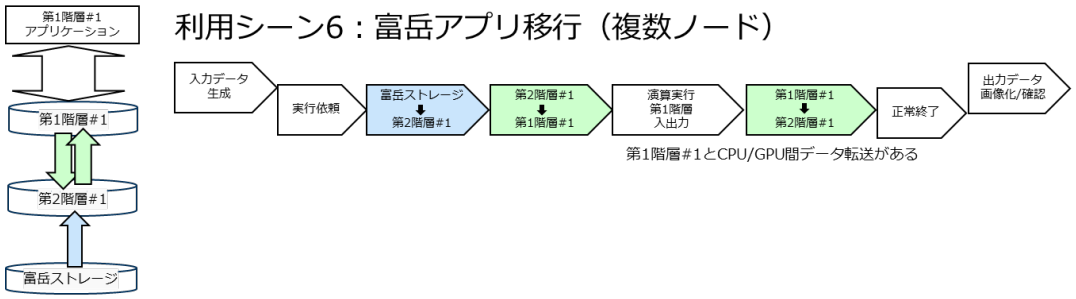


図 3-31 利用シーン 6

- 利用シーン 7
  - 外部ストレージ連携が必要なアプリケーションでの利用を想定する。入力出力データは外部ストレージ上から演算実行と並行して第 1 階層、第 2 階層に同期する。出力データは外部ストレージへデータ転送する。

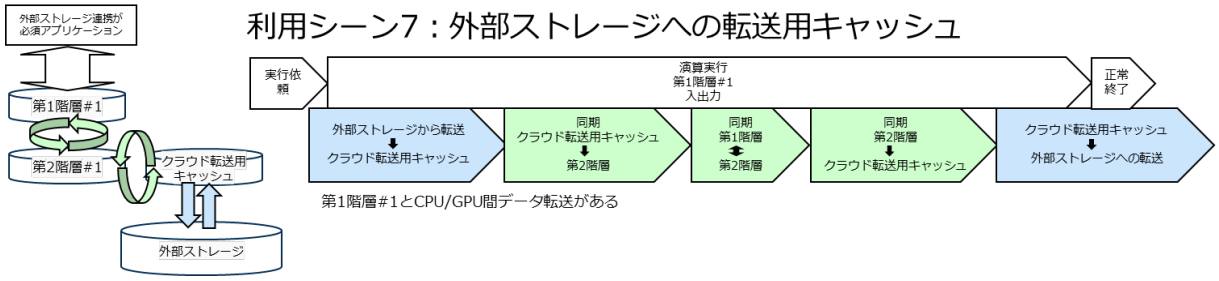


図 3-32 利用シーン 7

- 利用シーン 8
  - 「富岳 NEXT」の出力データのライフサイクル管理の想定である。ワークロードの完了後にアプリケーションの出力データを外部ストレージへデータ転送する。第 2 階層をデータアーカイブに利用され、ストレージ容量を逼迫しないように外部ストレージを活用するような運用やユーザの誘導が必要となる。

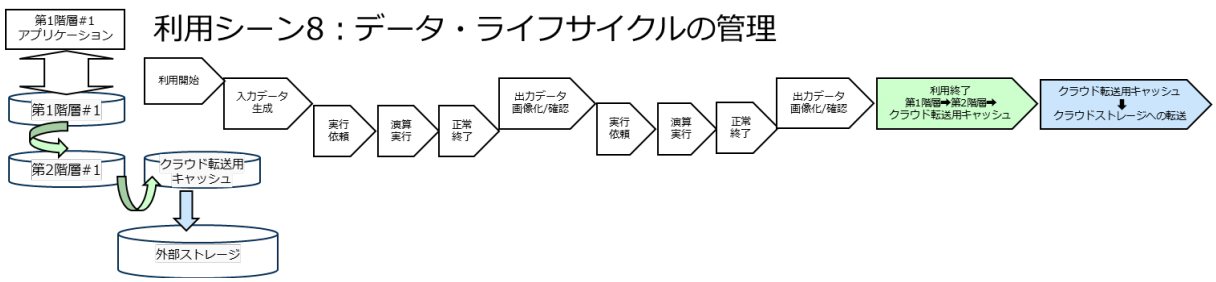


図 3-33 利用シーン 8

3.4.2.3 第 1 階層ストレージ

本節では、基本設計での第1階層ストレージの検討内容を記載する。3.4.2.2節で挙げた利用シーンを実現するための機能、FSの課題、および本検討会で上がった課題を組み込んでいる。

3.4.2.3.1 システム構成

本項では、基本設計での第1階層ストレージのシステム構成検討内容を記載する。

3.4.2.3.1.1 ハードウェア構成検討

本項では、第1階層ストレージのハードウェア構成検討内容を記載する。

第1階層ストレージはAll SSD構成を基本とし、利用するデバイスの検討結果については3.4.5.2.1項、第1階層全体のストレージ容量の検討結果については3.4.5.2.2項に記載している。ハードウェア構成としては、ノードローカルストレージとニアノードローカルストレージの2種類を検討対象とする。

ノードローカルストレージは計算ノードに搭載されたSSDを利用する方法である。計算ノードには複数のSSDが搭載されることが考えられるため、それらをどのように利用者に見せるかの検討が必要となる。ノードローカルストレージとして考えられる利用イメージとしては以下が挙げられる。

- 利用イメージ 1
  - 計算ノードに搭載された各 SSD に対して各々ファイルシステム（例：ext4、XFS など）をマウントして利用する。各デバイスの性能を活かすことができるが、冗長化はされていないのでデバイス故障の場合には対応できず可用性には欠ける。利用イメージ 1 を図 3-34 に示す。

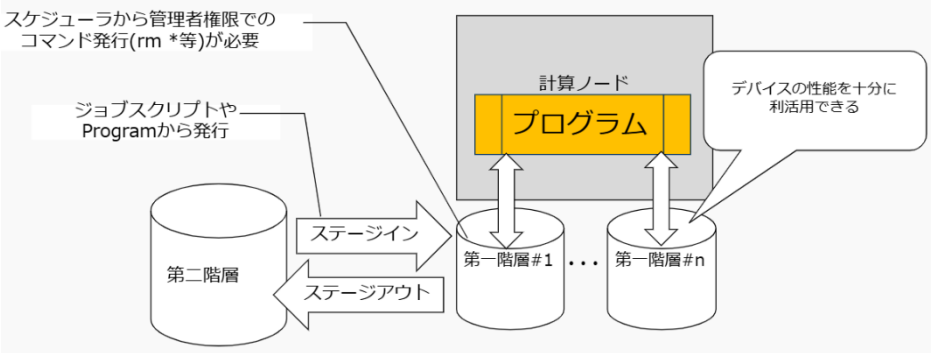


図 3-34 ノードローカルストレージ利用イメージ 1

- 利用イメージ 2
  - 計算ノードに搭載された各 SSD をファイルシステムのソフトウェア RAID によりまとめて利用する。デバイス故障の場合も対応でき可用性が向上するが、性能面では RAID によるオーバーヘッドを考慮する必要がある。利用イメージ 2 を図 3-35 に示す。

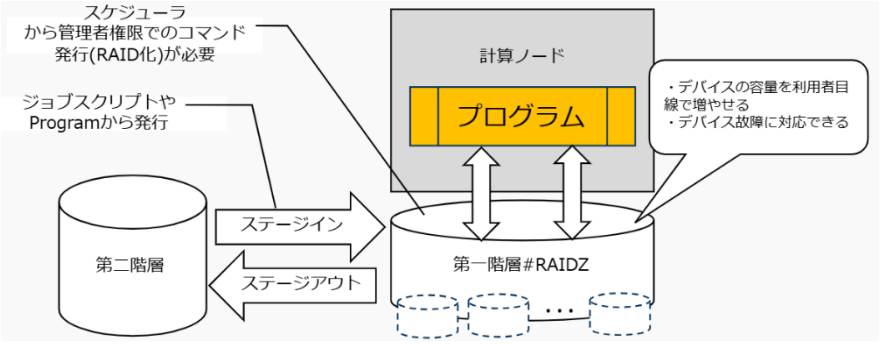


図 3-35 ノードローカルストレージ利用イメージ 2

ニアノードローカルストレージは他のノードに搭載されたSSDを利用する方式である。考えられる利用イメージとして以下が挙げられる。

- 利用イメージ 1
  - 2 つの計算ノード間で同一 SSD を PCIe で接続し、共有利用する。ノード故障時にも対となるノードで運用できるため可用性は向上するが、ハードウェア構成が複雑でコストが増加することや、運用が煩雑になることが予想される。また可用性の確保についても、アプリケーションで Erasure Coding を用いて対処することもできるため、ハードウェアレベルで実現することによる優位性は低いと考える。
- 利用イメージ 2
  - 計算ノードとは別のストレージ専用ノードを用い、各計算ノードからネットワーク経由で利用する。計算ノード障害の影響を受けないため運用安定性は向上するが、専用ノードが必要となりコストが増加することが予想される。
- 利用イメージ 3
  - ジョブ内で利用している計算ノードのローカルストレージをネットワーク経由で共有利用する。構成はシンプルとなりコスト面で優位だが、ジョブ内共有のためスケジューラーからの制御方法など検討項目は多くなることが予想される。

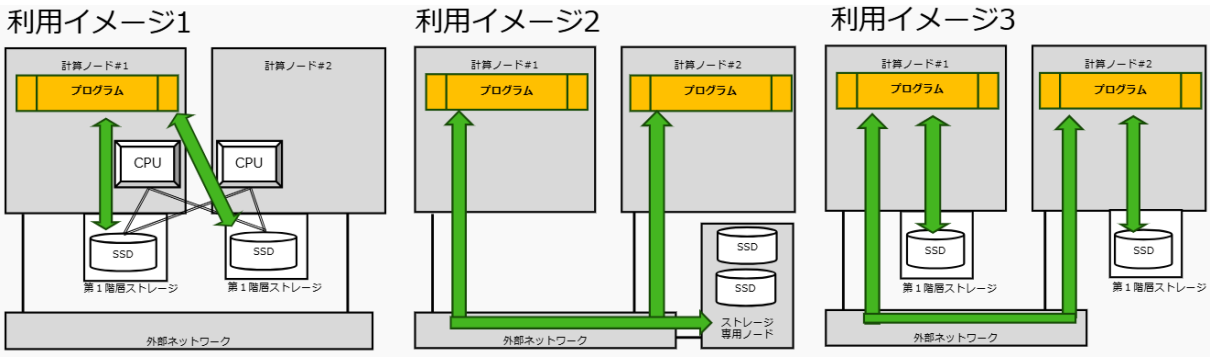


図 3-36 ニアノードローカルストレージ利用イメージ

基本設計では、ノードローカルストレージ・ニアノードローカルストレージの利用イメージと構成パターンの検討を実施した。各パターンにおいては、性能・コスト・運用性などの観点で長所・短所があり、特にニアノードローカルストレージにおいては構成面の複雑化や検討項目の増大などが想定されるため、現時点ではノードローカルストレージが有

力であると考えられる。詳細設計で各ベンダからの情報をもとに、より詳細な比較検討によってハードウェア構成を決定する。

参考として、ニアノードローカルストレージについて各パターンの見解の詳細を表 3-26に示す。

表 3-26 ニアノードローカルストレージの各利用イメージに対する見解詳細

区分	利用イメージ 1 たすきがけ PCIe 接続	利用イメージ 2 ストレージ専用ノード	利用イメージ 3 ジョブ内共有
構成上の特徴	・対向ノードの第 1 階層ファイルシステムを直接参照するため、メタデータ保護/データ整合性確保には物理ジャーナルが必須。	・専用ノードにより保護方式を柔軟に選択可能（レプリケーション/Erasure Coding 等）。	・ジョブ内でノード間の第 1 階層 SSD を相互利用する際、単純なリモートブロックアクセスだけでよいのか、Erasure Coding 等を適用して耐障害性を高めるのか構成判断が必要。
	・物理ジャーナル領域が SSD 容量を圧迫。	・スケールアウト構成によりノード単体障害の局所化が容易。	・計算ノード障害時に第 2 階層へ自動退避（Swap-out）してリカバリを行う設計も選択肢。
前提条件・制約	・物理ジャーナルにより SSD TBW 消費が大きい。	・専用ノード数/ネットワーク帯域で性能調整が可能。	・共有対象がジョブ単位。
	・PCIe 直結ゆえにノード配置変更やリプレイス時の制約が強い。	・ジョブ特性に応じた帯域設計が必要。	・Swap-out 多用時はメタデータ管理が複雑化。
性能・運用上の懸念	読み込み性能は実効で約 1/2 に低下する。	・CPU 負荷/レイテンシ増大し、外部 NW 負荷増による性能揺らぎあり。	・CPU 負荷/レイテンシ増大し、外部 NW 負荷増による性能揺らぎあり。
	ローカルと同様に外部 NW 負荷による性能揺らぎを抑制。		・Swap-out の多用で第 2 階層の負荷ピークが発生。
総合見解	・整合性は高いが、スケーラビリティ・柔軟性が大きく制限される。	・データセンター設計と親和性が高く、運用安定性も良い。	・最も柔軟だが方式選択で性能・可用性が大きく変動。
	・SSD 消費や性能低下を許容する必要。	・専用ハードが存在する場合は有力な選択肢。	・実運用設計で検討すべき項目が最多。

3.4.2.3.1.2 ファイルシステム構成検討

本項では、第1階層ストレージのファイルシステム構成検討内容を記載する。

- 基本構成
  - ファイルシステム種別
    - 第 1 階層ファイルシステムとしては計算ノードローカルなファイルシステム（例：ext4、XFS など）の他、複数の計算ノードからの SSF アクセスに対応するため、共有ファイルシステムも求められる。
  - 対応インターコネクト種別
    - 計算ノード間のインターコネクトの種類としては InfiniBand または Ultra Ethernet が想定される。共有ファイルシステムの場合、ノード間通信を行うためにこれらのインターコネクト技術への対応が求められる。
  - 構成モデル
    - ファイルシステムの構成要素としてデータおよびメタデータを格納するデバイス・ノードとそれらを管理するサービスを想定する。共有ファイルシステムの場合、計算ノードの他に管理用ノードの有無が構成に影響するため、構成要素を明確にすることが求められる。

- 仮想化対応性
  - 計算ノード環境としてコンテナ/仮想マシンなどの仮想化環境も想定される。ファイルシステムでは、仮想化環境においても第 1 階層ストレージへアクセスする方法を提供することが求められる。また、第 1 階層を共有ファイルシステムで構成する場合に、オーケストレーションシステムとの連携を選定時に考慮する必要がある。
- データ配置/メタデータ管理
  - 共有ファイルシステムの場合、データおよびメタデータを複数ノードに分散して格納することが想定される。その際に、ファイルシステムの構成要素として計算ノード間で相互に通信を行うためのコンポーネント（例：Remote Procedure Call など）が求められる。
  - 「富岳」では第 1 階層ストレージの SSD 搭載不足や、コネクション形成に必要なメモリ容量などが考慮されていなかったため、I/O 性能限界となる課題（STK-008/STK-013）がある。そのため、必要となる SSD 数やメモリ容量などの計算ノードへの要件についても明確にすることが求められる。

3.4.2.3.2　期待される機能

本項では、第1階層ストレージの運用管理上で期待される機能検討内容を記載する。

3.4.2.3.2.1　継続安定性・信頼性

本項では、ストレージシステムとして継続的に安定性・信頼性（データインテグリティ）を確保するために期待される機能を記載する。

- 冗長化・耐障害性

  安定して継続運用をする上で、ファイルシステムレベルでの冗長化・耐障害性の機能が求められる。冗長化・耐障害性として以下の 5 つの要素が考えられる。
  - 保護方式
    - ディスク障害発生時においてもデータの損失を防ぐための機能が求められる。例として、ソフトウェア RAID や複数のディスクにデータブロックとパリティブロックを分散書き込みする Erasure Coding が考えられる。また、ストレージ装置の媒体やインターフェース伝送経路におけるデータ整合性の喪失や、T10 PI のようなエンドツーエンドのデータ完全性の保証も選定の項目に挙げる。
  - メタデータ冗長化
    - メタデータサーバーを必要とする共有ファイルシステムの場合、メタデータサーバーの障害発生時にも運用継続するための機能が求められる。例として、メタデータサーバーを 2 台用意し、2 台とも常時稼働状態とする Active-Active 、または 1 台を稼働状態、1 台は待機状態とする Active-Standby の HA（High Availability）構成が考えられる。
  - データ冗長化
    - 共有ファイルシステムの場合、データを格納するストレージサーバーの障害発生時にも運用継続するための機能が求められる。例として、同一データを複製して複数のストレージサーバーに保管する方法が考えられる。
  - ネットワーク冗長化
    - 共有ファイルシステムの場合、ネットワーク障害発生時にも運用継続するための機能が求められる。例として、複数のネットワークパスを利用して同時に通信する方法が考えられる。
  - ノードフェイルオーバー

- ✧ ニアノードローカル構成時の共有ファイルシステムの場合、冗長化構成において障害発生時に自動で待機系のノード切り替わり、運用継続できることが求められる。また、障害検知、ノード切り替え、リカバリ処理による復旧時間が短いことが求められる。冗長化構成によっては、ノード切り替え後に負荷上昇と性能劣化が発生することが考えられる。

- データ削減
  - 「富岳」では、ストレージ容量が十分ではなく、研究成果創出の妨げになっている課題がある（STK-007）。そのため、格納するデータ量を削減するための機能が求められる。例として、データの圧縮や重複排除が考えられる。
- 負荷予兆機能
  - サーバ高負荷によるノードダウンやレスポンス低下などを防止するため、負荷状況を監視することで事前に検知・対処することが求められる。例として Grafana/Prometheus のような監視ソフトウェアと連携する方法が考えられる。
- 多数プロセス同時アクセス対策
  - 「富岳」では、多数プロセスが同時にファイル open/close を実行した際に、一部プロセスの応答時間が遅延する課題がある（STK-003）。そのため、多数のプロセスからメタデータアクセスがあった場合でもレイテンシを低く抑えることが求められる。
- Quota
  - 特定のユーザがストレージ容量を占有することを防ぐため、Quota の機能が求められる。例としてユーザ/グループ/プロジェクトの単位で Quota を設定可能とすることが考えられる。
- QoS
  - 計算ノード内において、特定のユーザがストレージへの I/O リクエストを占有することを防ぐための機能が求められる（ノード QoS）。また、複数ジョブ実行時に、特定ジョブが I/O を占有することを防ぐことが求められる（サーバ QoS）。

3.4.2.3.2.2　セキュリティー・データ保護

本項では、ストレージシステムとしてセキュリティの確保及びデータを保護するために期待される機能を記載する。

- アクセス制御/認証
  - ファイルやディレクトリに対する他ユーザやジョブからのアクセス制御の機能が求められる。例として POSIX ACL などへの対応が考えられる。
  - 共有ファイルシステムの場合、サーバへの不正なアクセスを防ぐため、信頼できるクライアントのみがサーバへアクセスできるような機能が求められる。例として Kerberos 認証などへの対応が考えられる。
- 暗号化
  - 転送中データおよび保管データについて盗聴・改ざんを防ぐための暗号化が求められる。例として TLS による通信経路の暗号化や、AES によるデータの暗号化が考えられる。
- 鍵管理
  - データ暗号化のための暗号鍵を安全に管理するための鍵管理システムとの連携が求められる。鍵管理システムの例として Vault などへの対応が考えられる。

3.4.2.3.2.3　将来性・拡張性領域

本項では、2030年を見据えたストレージシステムを選定するために将来性や拡張性の観点として考えられる項目を記載する。

- 新技術適用性
  - ZNS（Zoned Namespase）、SCM（Storage Class Memory）などの新技術の適用検討が求められる。
- ディスク技術
  - NAND フラッシュメモリーの種類（例：TLC/QLC/PLC など）やフォームファクタの適用検討が求められる。
- 技術的優位性を有する機能
  - ファイルシステム独自の強みとなる機能の有無が挙げられる。

3.4.2.3.2.4　制約・運用リスク

本項では、ストレージシステムを運用する上で機能上の制約やリスクとなり得る項目を記載する。

- 制約・注意点
  - ファイル数上限などの運用上の制限が挙げられる。
- エラー検知・再構築
  - エラー発生時の再構築・自動修復の機能有無や速度が挙げられる。
- その他のアーキテクチャへの要件・制約
  - 対応 OS 種類の制約や、外部機器の要否などが挙げられる。

3.4.2.3.3　システム要求性能の検討

本項では、第1階層ストレージの要求性能の検討内容を記載する。

第1階層ストレージの性能は計算ノードに搭載するデバイス（SSD）の性能および台数とファイルシステム性能に依存する。基本設計では、要求性能を満たすために必要となる下記項目に対してベンダーヒアリングを行い、情報収集を実施した。詳細設計では、ストレージシステムに関する要件整理を行い、当該要件に見合うストレージシステムが導入されるよう、ストレージベンダーと協調しながらの設計を検討する。また、必要に応じて各ベンダから提供される情報をもとに性能見積を行い、提案されたストレージシステムについて妥当性の検証および比較検討ができるよう整理する。

- スケーラビリティ
  - 「富岳」では、メタデータのスケーラビリティで以下のような課題がある。「富岳 NEXT」においてもアクセスする計算ノード数や扱うファイル数が増えた場合にもメタデータ性能スケーラビリティを確保することが求められる。
    - ✧ 同一ディレクトリにアクセスするノード数やファイル数が増大することにより性能低下や IO エラー頻度が上昇する課題（STK-002）
    - ✧ 大量ファイル作成時にメタデータサーバーの負荷が上昇し、全系規模までスケール出来ない課題（STK-004）
    - ✧ メタデータサーバーへの負荷集中による I/O エラーによってジョブが失敗または再実行する課題（STK-005）
  - 計算ノードの段階導入を見据えて、目標性能は各段階に応じて評価できる方式の検討が求められる（性能測定方針については 3.4.4.1 節を参照）。
- パフォーマンス特性
  - 選定の観点として、IO500 などのパフォーマンス実績の有無について確認することが求められる。
- ファイルシステム性能とデバイス性能

- システム要求性能及び想定デバイスでの推定性能については 3.4.5.2.2 項に記載する。ファイルシステムを介した場合のオーバーヘッドを考慮した場合でも、要求性能を満たすファイルシステムを選定することが求められる。

- デバイス依存性能
  - NVMe SSD、SCM（Storage Class Memory）など、要求性能を満たすデバイスを選定することが求められる。
- 並列 I/O
  - 共有ファイルシステムの場合、複数計算ノードからの並列ジョブに対応するため MPI-IO への対応が求められる。また、並列ライブラリとして HDF5/NetCDF の利用が想定されるため、これらについても対応が求められる。
  - 「富岳」では、MPI-IO の非同期 I/O のサポートが課題として挙がっている（STK-006）。MPI-IO の非同期 I/O については、ファイルシステムでの対応要否を含めて詳細設計で検討を実施する。
- 一貫性緩和
  - 厳格な POSIX 準拠の一貫性の他、性能を重視する場合に用いるための一貫性を緩和したモードなどが求められる。
- 代表的な性能計測・観測値
  - 性能についてはメタデータ性能（IOPS）、シーケンシャルアクセス性能（バンド幅）の測定が求められる。測定手法の検討については 3.4.4 章に記載する。
- SSF 性能の強さ
  - 「富岳」では SSF アクセス時に高並列ジョブでロック獲得オーバーヘッドによる性能低下が課題として挙がっている（STK-001）。そのため、高並列時でも SSF アクセス性能を確保するための機能（例：一貫性を緩和した I/O など）が求められる。
- GPUDirect Storage 対応
  - GPU からの I/O 高速化をする場合は、ファイルシステムでの GPUDirect Storage への対応が求められる。GPU による第 1 階層ストレージの利用方法などについては詳細設計以降で検討する。
- 非共有型アーキテクチャ(DASE)の有効性
  - VAST Data Platform に採用されている、コンピュートとストレージを分離した「シェアード・エブリシング(DASE)」構成を、大規模 AI 環境における重要な技術として注視する。
  - メタデータ競合を物理的に排除しつつ、性能と容量を独立してスケーリングさせる特性を注視し、単一名前空間でのエクサバイト級拡張の実現性を確認していく
- 分散メモリ統合と超高速給弾技術
  - WEKA-IO が提供する「Augmented Memory Grid」技術により、数千ノード規模で分散メモリを統合し、TB/s 級の超高速データ供給能力（給弾能力）を確保する手法なども詳細設計で確認していく
- 低遅延ホットデータパスの確保
  - Scality（RING XP）等に見られるマイクロ秒レベルの応答性能は、AI 推論ワークロードにおけるホットデータパスとして極めて有用であると考えられ、詳細設計においてその構成案を精査していく

3.4.2.3.4　ファイルシステム候補選定

基本設計では、3.4.2.3.1項から3.4.2.3.3項までの各項で第1階層ファイルシステムとして求められる要素を抽出した。詳細設計においては、各ベンダの情報をもとに、これらの要素を満たすファイルシステムを選定する。すべての要素を満たすことが難しい場合は、各ファイルシステムの機能開発による実現可否の検討もしくは複数のファイル

システムを組み合わせていくことによる実現可否の検討などを詳細設計で行っていく。

また、「富岳」においては、持続的なソフトウェア開発体制が不足しており、運用後の継続的な改修・改善に対する課題（STK-011、STK-012）があるため、選定基準の観点に含める必要がある。例として、OSSであればコミュニティの活動状況、ISVであればベンダの保守/サポート体制などが挙げられる。

3.4.2.4 第 2 階層ストレージ

本節では、基本設計での第2階層ストレージの検討内容を記載する。3.4.2.2節で挙げた利用シーンを実現するための機能、FSの課題、および本検討会で上がった課題を組み込んでいる。

3.4.2.4.1 システム構成

本項では、基本設計での第2階層ストレージのシステム構成検討内容を記載する。

3.4.2.4.1.1 ハードウェア構成検討

第2階層ストレージは全計算ノードで共有するストレージシステムを想定している。ストレージシステムは外部ストレージ装置と、それを管理するストレージサーバーで構成する。ストレージサーバーを複数ノード準備することで、第2階層のストレージ容量、バンド幅、IOPSを達成する。ストレージサーバーの必要台数については詳細設計で検討する。

第2階層ストレージの利用方法として、全体で1つのボリュームとして利用するのではなく、仮想的に分割して利用する構成も考えられる。その場合、そのうえで性能的に問題がないか、耐障害性を担保できるかを検討する必要がある。具体的な実現方法については詳細設計で検討する。

第2階層ストレージで利用するデバイスの検討結果については、3.4.5.2.1項に記載している。また、第2階層ストレージ全体の容量、バンド幅、IOPSの検討結果については3.4.5.2.2項に記載している。段階導入の検討結果については3.4.5.2.4項に記載している。

第2階層ストレージで利用するデバイスの選定の検討から、SSDの書き込み性能は読み込み性能より低くなることが想定される。そのため、容量と性能のバランスから性能要件の書き込み性能を満たすことが難しい可能性がある。読み込み性能をベースとし要求性能の見直しが必要となるため詳細設計以降で検討する。

3.4.2.4.1.2 ファイルシステム構成検討

本項では、第2階層ストレージのファイルシステム構成検討内容を記載する。

- 仮想化対応性
  - 計算ノード環境としてコンテナ/仮想マシンなどの仮想化環境も想定される。ファイルシステムでは、仮想化環境においても第 2 階層ストレージへアクセスする方法を提供することが求められる。
- データ配置/メタデータ管理
  - データの配置方法として、ストライピングによって複数のストレージサーバーに並列配置することが考えられる。またメタデータの管理については単一のメタデータサーバーで管理する方法や、複数ノードに分散して管理する方法などが考えられる。これらの方式が構成に影響するため、方式を明確にすることが求められる。
  - 「富岳」ではコネクション形成に必要なメモリ容量などが考慮されていなかったため、I/O 性能限界となる課題（STK-008）がある。そのため、メモリ容量も含めてメタデータサーバー及びストレージサーバーのハードウェア要件についても明確にすることが求められる。

以下の項目は、3.4.2.3.1.2項の第1階層の記載と共通である。

- 基本構成
  - 対応インターコネクト種別
  - 構成モデル

3.4.2.4.2　分割導入

第2階層ストレージシステムを導入するにあたり、ストレージシステムの全量を一括して導入するのではなく、ストレージサーバーの筐体などを2回以上に分けて段階的に導入することを分割導入と呼ぶ。また、ストレージシステムの一部を段階的に更新することをローリング型と呼ぶ。

検討を重ねた結果、課題があるため具体的な解決策や導入方式は、詳細設計で実施する。

検討事項

- 分割導入
  - 予算
    要求仕様を満たすストレージシステムを用意するための金額と予算が乖離した場合は用意できた規模に応じた性能やストレージ容量となり要求仕様以下になりえる。
  - 導入時期/設置場所/移設
    「富岳」のストレージシステムのデータ移行等で「富岳 NEXT」のストレージシステムを先行導入する場合は導入時期および、どの建屋に設置し、場合によっては新たな建屋に移設を行う必要がある。
    また、「富岳」のストレージシステムがいつまで利用可能であるによっては一括導入も視野に入るが、オーバーホールなどが必要であり現時点ではいつまで利用可能かは未定である。
  - ファイルシステムの切り出し方
    分割導入の場合は一括してストレージシステムの全量が設置されないため、ユーザが利用できるファイルシステムとして、段階的に導入される筐体等からどのようにファイルシステムを切り出すかの考慮が必要であり、現時点では案が 4 つある。
    なお、ファイルシステムによっては増設となる筐体を既存ファイルシステムに組み込むことができない場合も想定されるため、留意が必要である。

導入する筐体ごとにファイルシステムを切り出す案

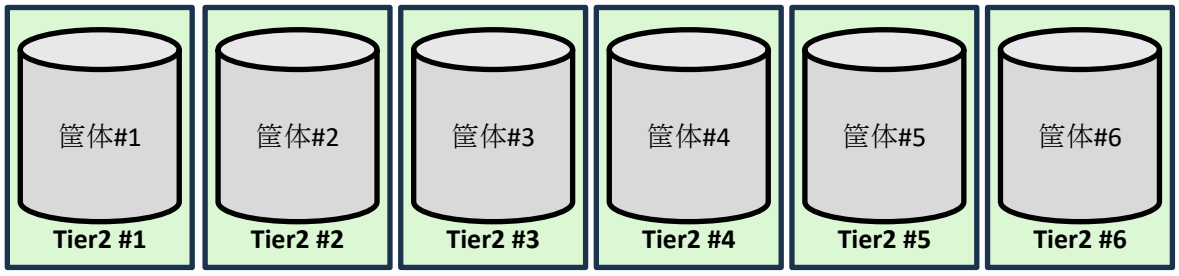


図 3-37 案 1 筐体別分割

分割する効果として、「富岳」でも分割されているとおり、１つのファイルシステムが停止しても、他ファイルシステムが稼働しているため、運用継続が可能となる。

導入する筐体をファイルシステムに増設しながらファイルシステムを切り出す案

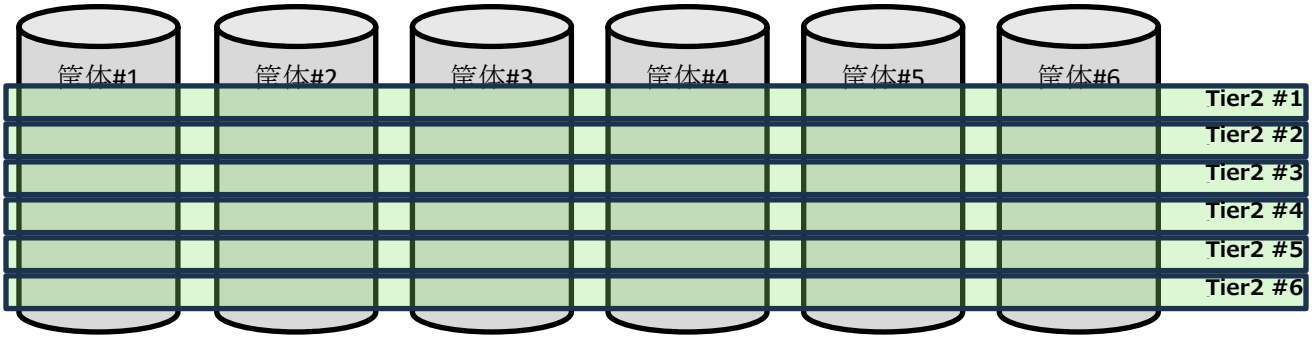


図 3-38 案 2 筐体跨り分割

案1と案 2 の組み合わせでファイルシステムを切り出す案

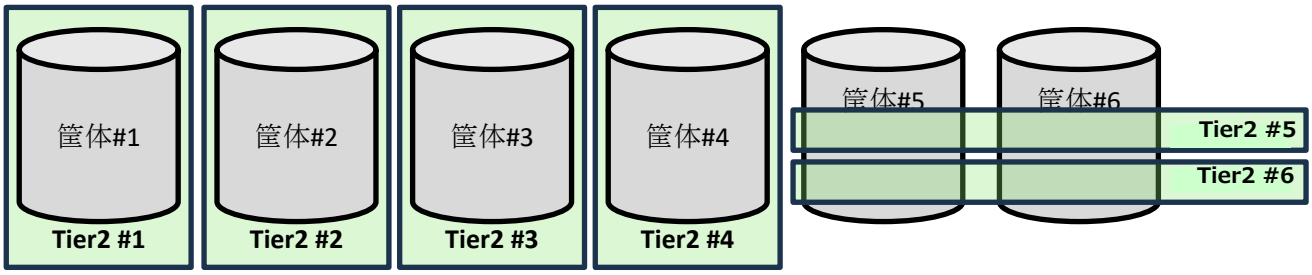


図 3-39 案 3 ハイブリッド

ユーザが利用する大規模 Tier が一つ、小規模テスト用 Tire が複数、全規模テスト用に全筐体を跨ぐ Tier が一つである案

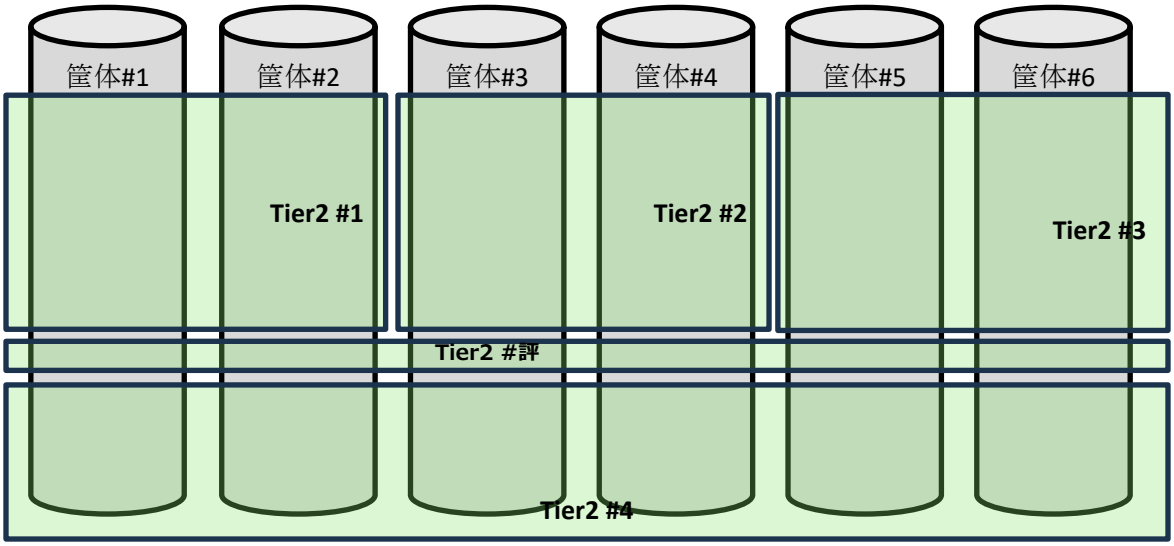


図 3-40 案 4

- ローリング型
  - 総性能

    段階的に更新することで、ストレージシステムの全量が揃わない期間が想定され全体性能の測定ができない場合がある。ローリング型更新の計画遅延や運用上の占有時間の制約が生じた場合にも同様に全体性能の測定できない場合もある。

    また、ストレージサーバーなどが同一構成ではない場合、全体の性能が最も遅いコントローラー等に律速される可能性がある。

    なお、一括導入ではストレージシステムの全量が同一世代・同一構成となるため全体性能の測定の総性能が一度に測定できるが、部品性能の向上や部品単価の低下といった恩恵を 5～6 年間受けられない。
  - システムの運用開始点

    どの段階までストレージシステムがそろった時点（例えば総性能測定が可能な段階など）をシステムの運用開始点とするのかの考慮が必要である。

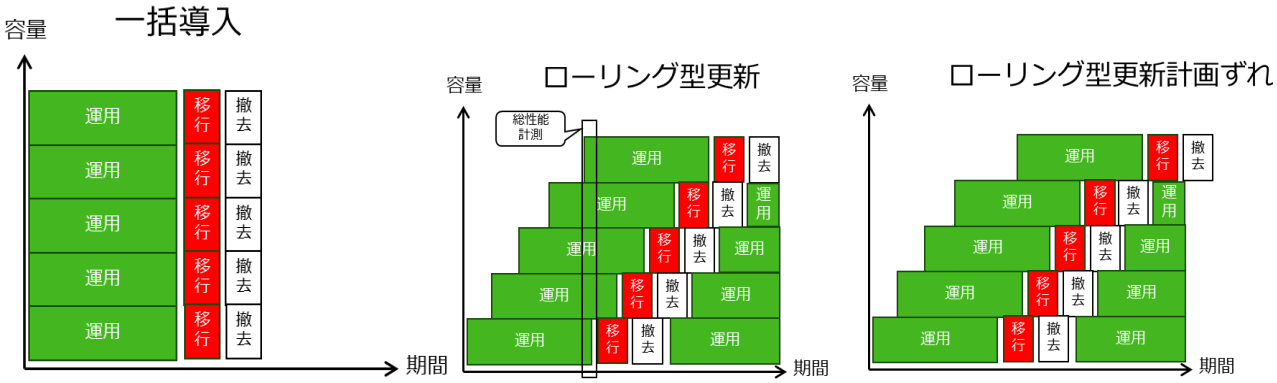


図 3-41 ローリング型

3.4.2.4.3　期待される機能

本項では、第2階層ストレージの運用管理上で期待される機能検討内容を記載する。

3.4.2.4.3.1　継続安定性・信頼性

本項では、ストレージシステムとして継続的に安定性・信頼性（データインテグリティ）を確保するために期待される機能を記載する。

- リバランス機能
  - 特定のストレージサーバーへのデータの偏りを防ぐため、ストレージサーバー間でデータ量の均一化を行うための機能が求められる。
- スナップショット
  - ファイルシステムの異常発生時に、リカバリを行うためのスナップショットを取得する機能が求められる。

以下の項目は、3.4.2.3.2.1項の第1階層の記載と共通である。

- 冗長化・耐障害性
- データ削減

- 負荷予兆機能
- 多数プロセス同時アクセス対策
- Quota
- QoS

3.4.2.4.3.2　セキュリティー・データ保護

本項では、ストレージシステムとしてセキュリティの確保及びデータを保護するために期待される機能を記載する。各項目は、3.4.2.3.2.2項の第1階層の記載と共通である。

- アクセス制御/認証
- 暗号化
- 鍵管理

3.4.2.4.3.3　将来性・拡張性領域

本項では、2030年を見据えたストレージシステムを選定するために将来性や拡張性の観点として考えられる項目を記載する。

- 構成柔軟性
  - 構成について要件別に複数のパターンを提案できる柔軟性が求められる。

以下の項目は、3.4.2.3.2.3項の第1階層の記載と共通である。

- 新技術適用性
- 技術的優位性を有する機能

3.4.2.4.3.4　制約・運用リスク

本項では、ストレージシステムを運用する上で機能上の制約やリスクとなり得る項目を記載する。各項目は、3.4.2.3.2.4項の第1階層の記載と共通である。

- 制約・注意点
- エラー検知・再構築
- その他のアーキテクチャへの要件・制約

3.4.2.4.4　システム要求性能の検討

本項では、第2階層ストレージの要求性能の検討内容を記載する。

第2階層ストレージの要求性能は計算ノードの規模に依存するため、要求性能に合わせてストレージ構成を考える必要がある。基本設計では、要求性能を満たすために必要となる下記項目に対してベンダーヒアリングを行い、情報収集を実施した。詳細設計では、ストレージシステムに関する要件整理を行い、当該要件に見合うストレージシステムが導入されるよう、ストレージベンダーと協調しながらの設計を検討する。また、必要に応じて各ベンダから提供される情報をもとに性能見積を行い、提案されたストレージシステムについて妥当性の検証および比較検討ができるよう整理する。

- スケーラビリティ
  - 3.4.2.3.3 項の第 1 階層に記載の内容に加え、第 2 階層ではストレージサーバー台数も性能に影響するため、ストレージサーバーのスケーラビリティも求められる。目標性能を達成するためのサーバ台数とコスト・容量のバランスを考慮して検討が必要と考えられる。

- ストレージサーバーの分割導入(3.4.2.4.2 項を参照)を見据えて、目標性能は各段階に応じて評価できる方式の検討が求められる（性能測定方針については 3.4.4.1 節を参照）。

以下の項目は、3.4.2.3.3項の第1階層の記載と共通である。

- パフォーマンス特性
- ファイルシステム性能とデバイス性能
- デバイス依存性能
- 並列 I/O
- 一貫性緩和
- 代表的な性能計測・観測値
- SSF 性能の強さ
- GPUDirect Storage 対応

3.4.2.4.5　ファイルシステム候補選定

第2階層では、ファイルシステムとハードウェアを含めたストレージシステムとして選定が必要となる。基本設計では、3.4.2.4.1から3.4.2.4.4項までの各項で第2階層ストレージシステムとして求められる要素を抽出した。詳細設計においては、各ベンダの情報をもとに、これらの要素を満たすファイルシステムおよびハードウェアを選定する。すべての要素を満たすことが難しい場合は、各ファイルシステムの機能開発による実現可否の検討もしくは複数のストレージシステムを組み合わせていくことによる実現可否の検討などを詳細設計で行っていく。なお、第2階層としてはオブジェクトストレージも候補に含めて検討を行う。

また、「富岳」においては、持続的なソフトウェア開発体制が不足しており、運用後の継続的な改修・改善に対する課題（STK-011、STK-012）があるため、選定基準の観点に含める必要がある。例として、OSSであればコミュニティの活動状況、ISVであればベンダの保守/サポート体制などが挙げられる。

3.4.3 ストレージシステム活用支援

本章では、ストレージシステムの活用を支援する技術として、第1階層/第2階層間のデータ転送、外部ストレージとの連携、およびストレージシステムの運用支援と性能分析のためのツールについて述べる。

3.4.3.1 第 1 階層/第 2 階層間のデータ転送

本節では、第1階層/第2階層間のデータ転送について、基本設計時点での調査・検討結果を記載する。なお、ジョブスケジューラやファイルシステムに依存する技術であるため、詳細設計以降で「富岳NEXT」が対応するジョブスケジューラやファイルシステムが決定した後に具体的な検討を行う。

3.4.3.1.1 転送技術・方法

データ転送技術として、主にファイルステージングを調査・検討し、透過的なキャッシュについても調査・検討を行った。ファイルステージングは、アプリケーションが使用する入力ファイルをアプリケーションの開始前に第2階層から第1階層に転送するステージインと、アプリケーションの終了後に出力ファイルを第1階層から第2階層に転送するステージアウトを行うことで、アプリケーションのファイルアクセスを高速な第1階層に限定し、アクセス時間の短縮を図る技術である。また、透過的なキャッシュとは、第1階層を第2階層のキャッシュとして利用することで高速なファイルアクセスを実現する技術である。第1階層/第2階層間のデータ同期をシステムが行うため、アプリケーションはキャッシュの存在を意識せずに利用可能となる。

データ転送方法としては、以下の4種類がある。それぞれの具体的な内容については3.4.3.1.3項に記述する。

- ジョブスケジューラ機能による転送
- ファイルシステム機能による転送
- 汎用コマンドによる転送
- その他の手段による転送

3.4.3.1.2 転送パターン

「富岳NEXT」で想定されるデータ転送のパターンを図 3-42に示す。ステージイン先として、計算ノードごとに割り当てられるノードローカル領域と計算ノード間で共有できる共有ファイルシステム領域を想定している。また、図中の1:NやSSF（Single Shared File） ⇔ FPP（File Per Process）は、階層間のデータ転送による入力ファイルの数や形式の変化を示している。(1)を例に取ると、第2階層上の1つのファイルをN台の計算ノードのノードローカル領域にステージインし（1:N転送）、ファイル形式がSSF（第2階層上のファイルにアクセスする場合）からFPP（第1階層上のファイルにアクセスする場合）に変化している。

なお、これらのパターンの内、ファイルの分割や結合を行う(4)、(6)、および(8)については、アプリケーション側で実現すべきパターンと考えられることから、対象外とし、(1)、(2)、(3)、(5)、および(7)を対象とする。

AIとシミュレーションを組み合わせるユースケースでは、それぞれで異なるアプリケーションセットが実行されるため、(2)が発生する可能性がある。AI観点でのパターンの検討については、GPUDirect利用の観点も含めて詳細設計で行う。(3)はプロセス（ランク）ごとに異なるファイルに入出力を行うパターンであり、チェックポイント/リスタートがその代表例である。チェックポイント/リスタートはアプリケーションからの指示でチェックポイントの作成（アプリケーションの実行状態の記録）と障害発生時などにチェックポイントからの再開を行う機能であり、アプリケーションと絡めた議論が必要であるため、詳細設計で検討を行う。

「富岳」では、プロセス間共通ファイルのステージングにllio_transferコマンドが利用されている。大気モデルの1つであるNICAMでは、プロセスごとに入力ファイルが分かれており、プロセスが配置されるノード近傍のストレージI/Oノ

ード（SIO）にファイルをステージングしたいという要求があるが、llio_transferコマンドの仕様では要求を満たすことができないという課題（STK-009）がある。図 3-42では、(3)がこの課題に該当する。

また、llio_transferコマンドには以下の課題もあり、詳細設計で検討する。

- ジョブスクリプトに llio_transfer コマンドを指定しなかったことが原因でアプリケーションが動作しなかったことがあり、llio_transfer コマンドの使い方をユーザに周知することが負担になっている。
- Python のファイルには 0 バイトのものがあるが、llio_transfer コマンドは 0 バイトのファイルを転送しない仕様のため、Python プログラムを実行できない。

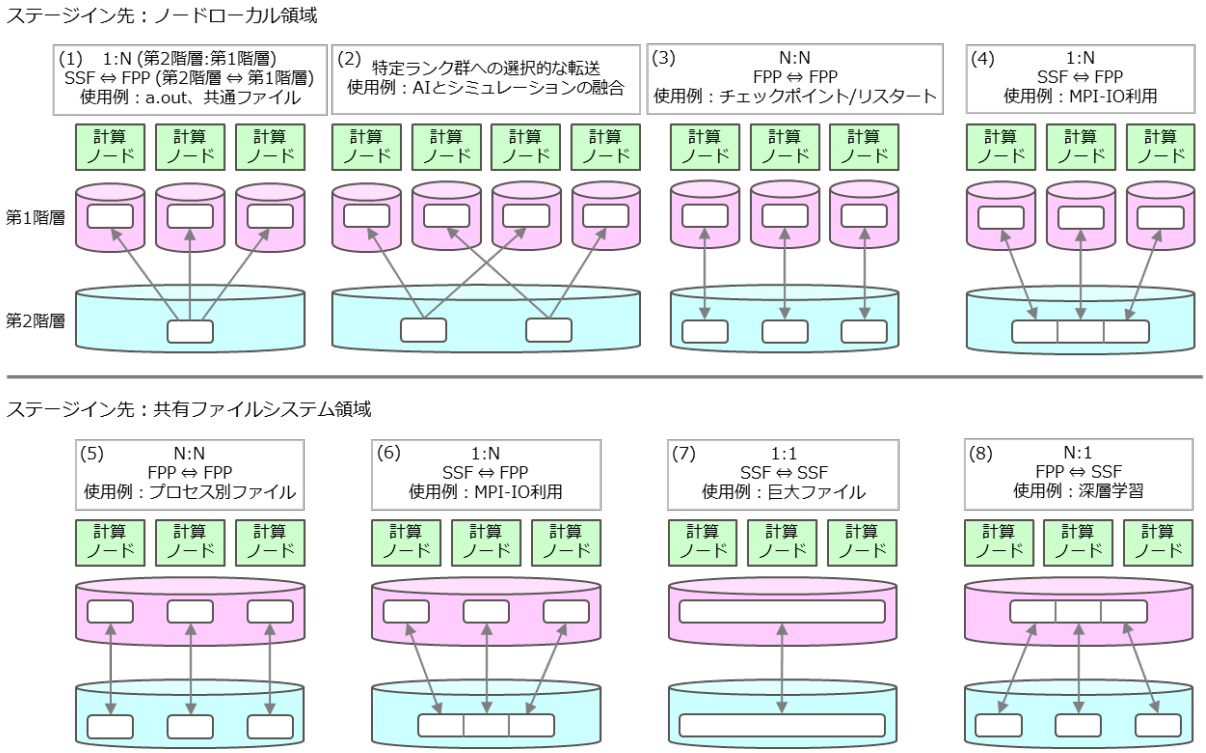


図 3-42 転送パターン

3.4.3.1.3　ソフトウェア調査

3.4.3.1.1項に記載したデータ転送方法について、基本設計時点で利用可能なソフトウェアを調査した。なお、3.4.3.1.2項に記載した各データ転送パターンのサポートについては、詳細設計以降で「富岳NEXT」が対応するジョブスケジューラやファイルシステムが決定した後に具体的な検討を行う。

1. ジョブスケジューラ機能による転送

(1) PBS Professional

- 概要

ジョブスケジューラの機能として、ファイルステージング機能を提供している。

- 指定方法

qsub コマンド（ジョブ投入コマンド）の-W オプションまたはジョブスクリプトの#PBS ディレクティブで指定する。

・ ステージイン

qsub -W stagein=<転送先パス>@<転送元ホスト>:<転送元パス>

または

#PBS -W stagein=<転送先パス>@<転送元ホスト>:<転送元パス>

・ ステージアウト

qsub -W stageout=<転送元パス>@<転送先ホスト>:<転送先パス>

または

#PBS -W stageout=<転送元パス>@<転送先ホスト>:<転送先パス>

- 備考

ステージング操作はジョブスクリプト実行ノードのみに限定されるため、図 3-42 転送パターンの(1)～(3)のような複数の計算ノードのノードローカル領域が転送先/転送元になるパターンに対応するには、別の仕組みが必要である。

(2) Slurm Workload Manager

● Burst Buffer Lua プラグイン

- 概要

ジョブスケジューラの機能として、ファイルステージング機能を提供しており、Burst Buffer の Lua プラグインを利用する。

- 指定方法

Lua プラグインが呼び出す Lua スクリプトにステージング処理を記述する。Slurm のソースコードに Lua スクリプトのテンプレートが提供されており、必要なパラメータ、各関数が呼び出されるタイミング、各関数の戻り値、各関数の記述例などが記載されている。

- 備考

Lua スクリプトにより、ステージング処理をカスタマイズ可能なため、様々な転送パターンに対応可能と考えられる。

● sbcast コマンド

- 概要

ジョブに割り当てられた各計算ノードにファイルを転送する。

- 指定方法

ジョブスクリプトに記述して実行する。

例) sbcast /global/a.out /local/a.out

第 2 階層の/global/a.out を各計算ノードの/local にステージインする。

- 備考

指定方法に記載したように、sbcast コマンドをステージインに利用できるが、各計算ノードから第 2 階層に出力ファイルを収集するといった、逆方向の使い方はできないため、ステージアウトには別の仕組みが必要である。また、ブロードキャスト用のコマンドであるため、図 3-42 転送パターンの(1)以外のパターンのステージングには適さない。

2. ファイルシステム機能による転送

(1) Lustre PCC (Persistent Client Cache)

- 概要

Lustre クライアントのローカルストレージ(SSD)をファイルシステムの透過的なキャッシュとして利用する。

- 指定方法

Lustre クライアントで以下のコマンドを実行する。

- lfs pcc attach コマンド

  指定したファイルをローカルストレージに関連付ける。これにより、当該ファイルがローカルストレージにキャッシュされ、読み書き可能になる。

- lfs pcc detach コマンド

  指定したファイルのローカルストレージへの関連付けを解除する。必要に応じて Lustre サーバにキャッシュがフラッシュされる。

- 備考

  厳密にはファイルステージングとは異なるが、ファイルステージングと同様に、アプリケーションのファイルアクセスを高速な第 1 階層に限定することで、アクセス時間の短縮を図ることができる。また、特定のルールに基づいてファイルをローカルストレージに自動的にキャッシュさせることも可能である。

3. 汎用コマンドによる転送

(1) cp

- 概要

  汎用のファイルコピーコマンド。

- 指定方法

  ジョブスクリプトに記述して実行する。

- 備考

  異なるノード間のファイル転送はできないため、図 3-42 転送パターンの(1)～(3)のような複数の計算ノードのノードローカル領域が転送先/転送元になるパターンに対応するには、pdsh コマンドや mpiexec コマンドなどの並列実行コマンドと組み合わせて実行する必要がある。

(2) scp/rsync

- 概要

  汎用のファイル転送コマンド。

- 指定方法

  ジョブスクリプトに記述して実行する。

- 備考

  ジョブスクリプト実行ノードから他の計算ノードにファイルを転送できるため、図 3-42 転送パターンの(1)～(3)のような複数の計算ノードのノードローカル領域が転送先/転送元になるパターンにも対応可能である。ただし、複数の計算ノードへの並列転送や複数の計算ノードからの並列転送はできないため、繰り返し実行する必要がある。

(3) pdcp/rpdcp

- 概要

  複数のノードに並列にファイルを転送できる汎用コマンド。

  - pdcp コマンド：複数のノードに並列にファイルを転送する。
  - rpdcp コマンド：複数のノードから並列にファイルを転送する。

- 指定方法

  ジョブスクリプトに記述して実行する。

- 備考

  ジョブスクリプト実行ノードから他の計算ノードにファイルを転送できるため、図 3-42 転送パターンの(1)～(3)のような複数の計算ノードのノードローカル領域が転送先/転送元になるパターンにも対応可能である。

4. その他の手段による転送

(1) mpiFileUtils

- 概要

  MPI ベースのファイル操作ツール。MPI のアプリケーションとして実行する。

  ツール例)

  • dbcast：各計算ノードに単一ファイルをブロードキャストする。
  • dcp：ファイルまたはディレクトリを再帰的にコピーする。
  • dsync：2 つのファイルまたはディレクトリを同期する。

- 指定方法

  ジョブスクリプトで mpiexec コマンドまたは mpirun コマンドに指定して実行する。

  • ステージイン(例)

    mpiexec -n 128 dbcast /global/a.out /local/a.out

    または

    mpiexec -n 128 dcp /global/a.out /local

    第 2 階層の/global/a.out を各計算ノードの/local に転送する。

  • ステージアウト(例)

    mpiexec -n 128 dcp /local/¥* /global

    各計算ノードの/local 内の出力ファイルを第 2 階層の/global に転送する。

- 備考

  大規模システムでの利用を想定して設計されているツールであり、ノード数やプロセス数に応じた効率的な転送を実現できる。ただし、各計算ノード上で同じ処理を実行するツールのため、図 3-42 転送パターンの(2)や(3)のような計算ノードごとや特定ランク群ごとに異なるファイルを転送したい場合には工夫が必要である。

3.4.3.2 外部ストレージ連携

外部ストレージは、データのアーカイブや異なるスパコンシステム間でのデータ共有などのために利用が想定される。本節では、第2階層ストレージと外部ストレージ間でのデータ転送などの連携について以下の項目に関して調査した内容を示す。

1. 第2階層ストレージとの連携方法
2. 第2階層ストレージとの転送方法

外部ストレージとの詳細な連携方法や転送方法については、「富岳NEXT」で対応する第2階層、および外部ストレージにも依存するため詳細設計以降で検討する。

また、富岳ではHPCI共用ストレージなどの外部ストレージとの連携において、データの移動頻度や速度不足に起因した転送負荷の課題がある（STK-010）が、この課題に対する具体的な対処方法についても詳細設計以降で検討を行う。

3.4.3.2.1　第 2 階層ストレージとの連携方法

第2階層ストレージと外部ストレージ間の連携は、双方にアクセス可能なサーバを用意し、そのサーバ上でデータ転送用のソフト/サービスを利用してデータ転送を行う方法や階層型ストレージ管理ソフトウェアを導入して管理するという方法が考えられる。

なお、データ転送サーバは、ログインノードや計算ノードとは分けるのが一般的であるため「富岳NEXT」でもその方がよいと考える。

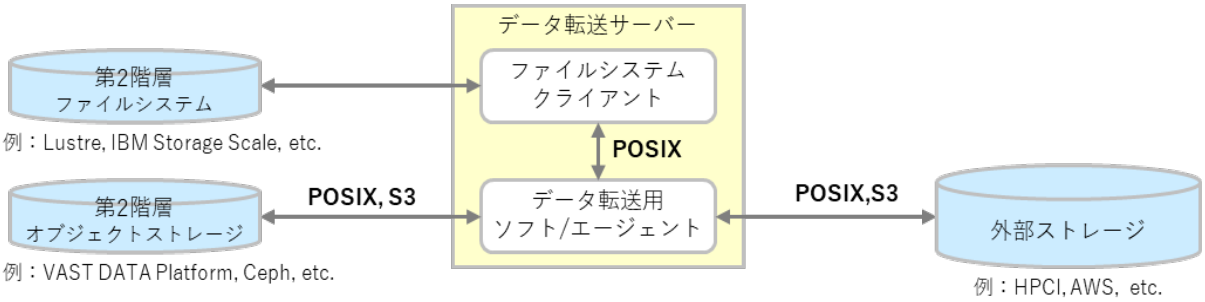


図 3-43 外部ストレージとのデータ転送

以下に連携対象の候補となる外部ストレージサービスを示す。どの外部ストレージに対応するかは詳細設計以降に検討する。検討にあたっては、各ストレージへアクセスする際の認証手段についても考慮する。

- HPCI 共用ストレージ
  対応 API：POSIX
- Amazon S3
  対応 API：S3
- Google Cloud Storage
  対応 API：S3（互換 API）、専用 REST API
- OCI Object Storage
  対応 API：S3（互換 API）、専用 REST API
- Azure Blob Storage
  対応 API：S3（要変換ツール）、専用 REST API

以下に外部ストレージとのデータ転送に利用できる転送用のソフト/サービスを示す。いずれのソフト/サービスも第2階層側へのアクセスにはPOSIXおよびS3のAPIが利用可能で、外部ストレージ側とのアクセスにはデファクトのAPIとなっているS3のAPIを利用することができる。

- Rclone
  オープンソースのコマンドラインプログラム。
  rsync、cp、mv、mount、ls、rm、cat などに相当するクラウド版のコマンドを備える。
- Globus
  シカゴ大学によって開発、運営されているデータ転送サービス。
  Web を利用した GUI とコマンドラインのインターフェースを持つ。
- AWS DataSync

AWS が提供しているデータ転送サービス。
AWS DataSync console を利用した GUI とコマンドラインのインターフェースを持つ。
AWS のストレージサービスとのデータ転送専用。

- Google Storage Transfer Service
  Google が提供しているデータ転送サービス。
  Google Cloud Console を利用した GUI とコマンドラインのインターフェースを持つ。
  Google Cloud Storage とのデータ転送専用。

「富岳NEXT」内部のストレージと外部ストレージを合わせて階層型ストレージとして管理する形態とした場合に利用可能と考えられる階層型ストレージ管理ソフトウェアを以下に示す。

- ScoutAM
- HPSS

3.4.3.2.2 第 2 階層ストレージとの転送方法

外部ストレージとのデータ転送は、自動で行う方法と手動で行う方法の2通りがある。

- 自動転送
  例えば Lustre の HSM 機能を利用することで設定した条件で自動的に外部ストレージとデー タ転送を行うことができる。
  他にもデータ転送サービスの中にはポリシーを設定し、特定の条件を満たした場合にデータ転送を行うことが可能なものがある。
- 手動転送
  ユーザが任意のタイミングで転送用のコマンドを実行してファイルデータを転送する方法。

以下に第2階層ファイルシステムとしてLustreを使用した場合の外部ストレージとのデータ転送イメージを示す。

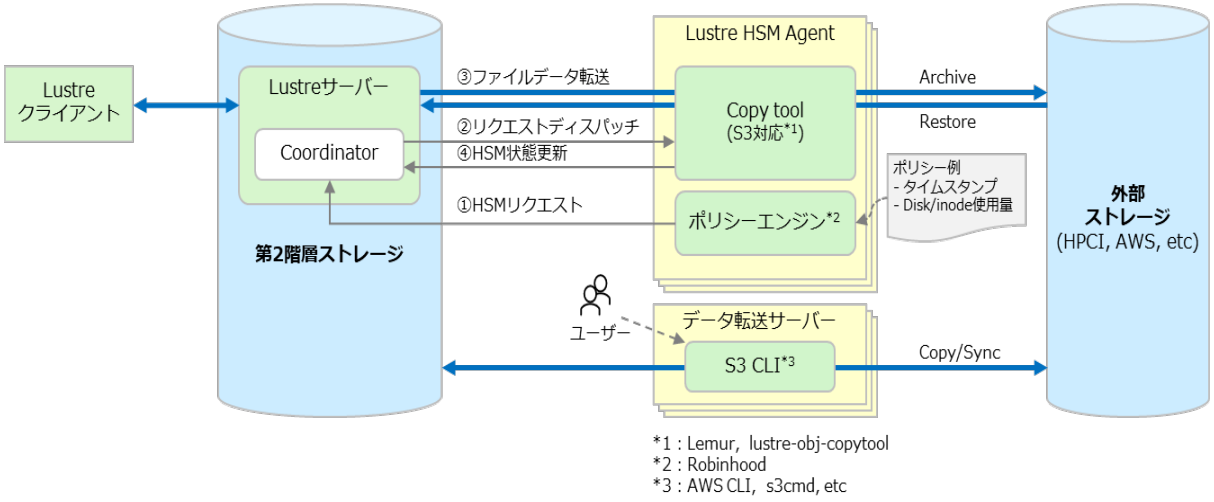


図 3-44 第 2 階層（Lustre）と外部ストレージ間でのデータ転送

図 3-44で示されているAgentやデータ転送サーバの種類や台数については詳細設計以降で検討する。図3-44内で使用しているツールの概要は以下の通り。

自動転送で使用

- Lemur
  Lustre 向けの HSM 用ツールキット。
- Lustre-obj-copytool
  Lustre の HSM 機能で外部のオブジェクトストレージ（S3 互換など）との間でデータをアーカイブしたり復元したりするためのツール。
- Robinhood
  大規模ファイルシステムのコンテンツを管理するための多用途ツール。
  Lustre ファイルシステム向けの高度な機能（OST またはプールごとのファイルのリスト/パージ、MDT の変更ログの読み取りなど）を実装している。

手動転送で使用

- AWS CLI
  コマンドラインシェルのコマンドを使用して AWS のサービスを操作できるツール。
- s3cmd
  Amazon S3 や、Google Cloud Storage、DreamHost DreamObjects など、S3 プロトコルを使用する他のクラウドストレージサービスプロバイダーでデータをアップロード、取得、管理するためのコマンドラインツールおよびクライアント。

さらにデータ管理とポリシー駆動型運用の高度化において以下の点に留意して詳細設計を進める必要がある。

- データ・プロベナンス（出所）管理の自動化
  - 学術界で実績のある iRODS について、最新の「Policy Composition」フレームワークを活用した、ペタバイト規模の科学データの Provenance 管理とライフサイクル自動化の柔軟性
  - 異種ストレージが混在する環境下においても、論理的な名前空間を一元管理し、国内外の研究機関とセキュアにデータフェデレーション（連携）を行う基盤としての適性
- メタデータ駆動型ガバナンスとコスト最適化
  - STARFISH Storage のようなメタデータ駆動型ツールによる、数十億ファイル規模の可視化と「Cost Control」機能の有効性
  - 膨大な非構造化データの重複検出やアクセスの可視化を通じて、ストレージコストを最適化しつつデータガバナンスを維持する運用モデル
- 解析ワークフローのインシトゥ（その場）化：
  - Intelligent Light（Kombyne）などの、超大規模シミュレーションの実行中にデータを移動させず、メモリ上で直接可視化・データ縮約を行う「インバスクロー（in transit）」技術の適用可能性

3.4.3.3 ストレージシステム支援及び性能分析ツール

3.4.3.3.1 期待される機能

本章では、ストレージシステムの運用支援および性能分析を実現するためのツールについて、調査・検討結果を報告する。これらのツールは、性能メトリクスの収集・可視化・解析を通じて、富岳NEXTのストレージシステムの健全性維持、ボトルネック特定、運用効率化を支援することが期待される。基本設計における検討の結果、以下の方針に基づき開発を進めることとした。

1. 選定したファイルシステムに付随する監視・分析機能を原則利用する。
2. 運用上不足する機能は、必要最小限の環境整備や補助ツール（例：性能データの連携スクリプト、カスタムダッシュボードなど）の開発で補完する。
   なお、補完対象項目はファイルシステム決定後、詳細設計で具体化する。

3.4.3.3.2　ツール候補選定

ストレージシステムの運用支援および性能分析を実現するため、現時点で利用可能な代表的な技術について調査を実施した。調査対象には、フラッグシップシステムで広く採用されている Lustre と、AIインフラ領域で採用が進む VAST Data Platform の関連ツールを抽出した。なお、本調査は一例であり、最終的な採用に際しては、対象とするファイルシステムの特性や運用要件に応じて、追加の評価・検討を行うことを前提とする。

1. VAST Management System (VMS)
   - 主な用途
     VAST ストレージの統合管理、性能監視、データフロー解析。
   - 特徴
     帯域幅、IOPS、レイテンシなどの性能指標、データ削減率やメタデータ使用率などの容量情報を即時可視化。複数テナント環境に対応し、各テナントのリソース使用状況を可視化。
   - 主要機能
     - GUI Dashboard および CLI / REST API によるアクセス
     - DataFlows（I/O 経路トレース）
     - Analytics（詳細メトリクス）
     - Top Actors（テナント占有状況）
     - VAST Catalog（メタデータのデータベース）
2. LustrePerfMon
   - 主な用途
     Lustre 環境の長期傾向監視、時系列分析、ダッシュボード可視化。
   - 特徴
     OSS スタック（collectd＋InfluxDB＋Grafana）を活用し、柔軟なダッシュボード構築が可能。時系列分析に強み。Lustre コミュニティで広く利用される標準的な監視基盤で、商用版も存在（例：ScalePOD Barreleye）。
   - 主要機能
     - Lustre プラグインによるメトリクス収集
     - InfluxDB で時系列データ管理
     - Grafana でダッシュボード表示（OST/MDT/OSS 単位、ジョブ単位）
3. MELT (Monitoring Extreme-scale Lustre Toolkit)
   - 主な用途
     超大規模クラスタでの広域監視、オンデマンド診断、根本原因分析。
   - 特徴
     拡張型 TBON（Tree-Based Overlay Network）構造により、複数ネットワークにまたがるスケーラビリティと低オーバーヘッドを実現。サーバー・クライアント双方を監視対象とし、I/O 状況を詳細に特定。
   - 主要機能

- サーバ/クライアント/ジョブ単位の計測、表示
- top コマンドライク表示による高負荷 I/O 特定
- CLI による操作、表示

4. LASSi
   - 主な用途
     ジョブ/アプリ単位の I/O 解析、競合検出、性能変動の把握。
   - 特徴
     独自の Risk/Quality 指標でジョブ単位の Read/Write 性能変動を MDS/OSS ごとに定量化し、ファイルシステムのスローダウンリスクやアプリ単位の I/O 効率を分析。スケジューラー連携をサポート。
   - 主要機能
     - I/O 統計からジョブ単位のプロファイル生成
     - ファイルシステムの異常や性能低下の検知、特定
     - 日次レポートの自動生成
     - CLI による情報取得
5. lltop / xltop
   - 主な用途
     ジョブ単位のリアルタイム I/O 負荷監視。
   - 特徴
     top コマンドライクな CLI で即時性に優れる。瞬間的な負荷把握に特化。シンプルな実装で、導入容易性が高い。
   - 主要機能
     - ジョブおよびホストごとの詳細な I/O 統計表示
     - 高度なフィルタリングとソート機能
     - CLI による操作、表示
6. LMT (Lustre Monitoring Tool)
   - 主な用途
     サーバ側（MDS / OSS / LNet）の負荷監視、履歴分析、障害解析。
   - 特徴
     Cerebro＋MySQL による履歴保存と ltop / lwatch での可視化。サーバ側の詳細監視と長期履歴管理に強み。
   - 主要機能
     - Cerebro プラグインでメトリクス収集
     - MySQL で履歴保存
     - CLI（ltop）と GUI（lwatch / lstat）で表示
     - 集計スクリプトで日次・月次レポート生成

以上の調査結果をまとめると、ツール選定における観点として以下が抽出された。

- ツールの特性と用途に応じた使い分け
  - CLI ベースのツール（lltop, xltop, MELT, LMT）
    軽量かつ即時性に優れ、瞬間的な負荷把握やノイズ源の特定、自動化に適する。

- GUI ベースのツール（VMS, LustrePerfMon）
  直感的な操作性、データの可視化、時系列分析、統合的な運用管理に優れる。API 連携により自動化の補完も可能。

- 多角的な情報収集と分析能力
  - メトリクス粒度と収集間隔の最適化
    情報収集の粒度に対する要求（例：平常時は粗粒度で低オーバーヘッド、異常時は詳細粒度）。
  - スケジューラー連携
    特定のジョブやアプリケーションが引き起こす I/O ボトルネックやノイズ源、その影響範囲を迅速に特定。
  - スケーラビリティと低オーバーヘッドの確保
    超大規模環境でも監視負荷を抑え、広域監視を実現。
- 複数ツール連携による相乗効果
  - 複合利用の検討
    単一ツールで全要件を満たせない場合、各ツールの強みを活かし、複数のツールを組み合わせて運用（例：リアルタイム監視＋長期傾向分析）。
  - 連携可能性の有無
    共通のデータ形式、API 連携などによる統合運用の容易性。
- 運用体制とコスト
  - 運用管理に必要な専門知識と構築コスト
    導入・保守にかかる負荷を評価。
  - 商用サポートの有無と範囲
    長期運用におけるリスク低減策。
  - コミュニティサポート
    OSS ツールの更新頻度や情報共有の活発度。
- 運用の自律化とセキュリティ・レジリエンス
  - AI 駆動による自律型ストレージ管理
    NetApp BlueXP や VAST に見られる、AI/ML モデルを用いたストレージヘルスの予測診断や、異常検知機能
  - ランサムウェア攻撃等に対する「自律的な防御・数分レベルでの高速復旧」を可能にするサイバーレジリエンス技術
  - 次世代暗号技術とデータ整合性
    将来的な脅威を見据え、P2P 暗号化を基盤とした**量子耐性暗号（QRS）**プロトコルの導入や、AI によるデータの不整合検知・自動修正機能

3.4.4 性能測定方針とベンチマーク手法

本章では、ファイルシステムの性能測定方針と性能測定の安定性を定量的に評価するためのベンチマーク手法について述べる。

3.4.4.1 性能測定方針

第2階層ストレージシステムを導入するにあたり分割導入が想定されており、目標性能を各導入段階で検討する必要がある。そのため、性能測定方針を以下とする。

- 目標性能は分割導入の各導入時点のストレージシステム規模をもとにバンド幅、IOPS を計算し、決定することとする。予算の都合などで全ストレージシステムの導入がされない場合にも同様とする。
- 分割導入の各導入時点でのファイルシステムはすべて同一であることとする。最初に導入されたストレージシステム規模の目標性能を達成すればよいなど性能評価については詳細設計以降で検討する。
- 性能要件にある計算ノードの全メモリは GPU のメモリ量を含まない CPU のメモリ量であり、現段階では読み込み性能のみである。

3.4.4.2 ベンチマーク手法

ファイルシステムのベンチマーク手法を述べる上で、以下を想定する。

- 第 1 階層のファイルシステム構成はローカルファイルシステムもしくは共有ファイルシステムとするかは検討中である。そのため、いずれのファイルシステム構成の場合でも定量的な評価が可能なように検討を行う。また第 1 階層は複数の計算ノードでの評価を行う。
- ファイルシステムの定量的な評価には複数回の測定を行い、ユーザ観点から最大値ではなく平均値を用いる。また、安定性を評価するために平均値からのばらつきが一定範囲内（例えば±10%）であることを確認する。複数回の実行回数やばらつきの範囲については詳細設計以降で検討する。
- ファイルシステムを利用するユーザは、単一のディレクトリ上でファイル作成などを行う傾向にあるため、ベンチマークにおいても単一ディレクトリに対して評価する。
- ファイルシステムの性能評価において、GPUDirect Storage のように GPU からファイルシステムに直接データ転送を行うことも想定される。また、GPU によるファイルシステム性能評価については詳細設計以降で検討する。なお、性能要件における「計算ノードの全メモリ」の定義は、GPU ではなく CPU に接続されているメモリとする。
- ジョブによっては他ジョブの外乱などにより一部のランクからのファイルの open/close が遅くなる場合がある。安定性を評価するために、複数のベンチマークを同時実行し、各ベンチマークの結果を評価するといった手段が考えられる。詳細については詳細設計以降で検討する。

表 3-27にファイルシステム性能を定量的に評価するベンチマークに利用される代表的ツールを調査した。また、富岳においてI/O性能が問題になっているアプリケーションについてベンチマークプログラムの準備されているものを示す。なお、GPUを用いたツール、および、アプリケーションによる評価方法については詳細設計以降で検討とする。

表 3-27 ファイルシステム性能を定量評価する代表的ツールおよびアプリケーション

ツール、アプリケーション名	評価内容
IOR	・I/O スループット性能を測定 ・GPU の I/O スループット性能を測定

mdtest	・メタデータ性能を測定
IO500	・ファイルシステムの総合的な評価
NVIDIA GPUDirect Storage（GDSIO）	・GPU の I/O スループット性能を測定
elbencho	・I/O スループット性能を測定 ・メタデータ性能を測定 ・GPU の I/O スループット性能を測定
Deep Learning I/O Benchmark（DLIO）	・Deep Learning の学習におけるストレージ I/O 性能を測定
Athena++	・アプリケーションによるベンチマーク ・FPP および SSF のファイル出力
R2D2	・アプリケーションによるベンチマーク ・MPI-IO による SSF のファイル出力
NICAM	・アプリケーションによるベンチマーク ・FPP のファイル出力

基本的な性能測定の考え方を以下に示す

- 多段階評価プロトコルの体系化

  ストレージの物理限界からアプリケーションの実効性能までを、以下の 5 段階（Phase）で段階的に明らかにする評価体系を行う 以下に基本的な例を示す

  Phase 1：物理通信限界の把握 (osu_micro_benchmarks)

  通信路（土管）の物理限界値を把握し、仮想環境下における物理応答への肉薄度を確認する。18-19μs レベルの低遅延や、理論値に近い帯域幅の維持を評価基準とする

  Phase 2：ストレージ単体基礎評価 (fio)

  チューニングが容易なシンプルなツールを用い、ストレージシステム自体の素のポテンシャル（Read/Write/IOPS）を、単一クライアント制約下での飽和点として確認する

  Phase 3：HPC ワークロード詳細評価 (IOR, mdtest, pfind)

  単一共有ファイル（SSF）への高並列書き込み競合（ロック競合）への耐性を評価する

  メタデータ性能においては、単一ディレクトリへの集中アクセス（ホットスポット）時のスケーラビリティ限界を、 files/sec 等の指標で定量化する

  pfind を用いた並列探索性能評価により、OS 標準の直列的な find に対する優位性を確認する

  Phase 4：AI 実効性能分析 (DLIO Benchmark)

  AI 業界標準の DLIO を用い、計算リソース（GPU）の稼働率を示す**AU（Accelerator Utilization）**を最重要指標として導入する

  ストレージからのデータ供給が計算機の「待ち時間（データストール）」に与える影響を定量化する

  Phase 5：高度シナリオ・安定性評価

  QoS 制御の有効性や、複数ジョブ・複数ワークロードが混在するコンテンション発生時の、実効性能の維持率を評価する

  留意点：

  Phase 2 で得られた「ストレージの潜在能力（fio 値）」と、Phase 4 で得られた「実アプリケーション性能（DLIO 値）」の間に生じる**性能乖離（オーバーヘッド）**を可視化する

試験の結果、ストレージ側の帯域に十分な余裕がある場合でも、実効性能がその数分の一（例：10%程度）に留まる「真のボトルネック」が、CPU 側のデータ前処理や並列制御にあることを特定する分析フローを確立

OS キャッシュの影響を排除するため、物理メモリ容量の 2 倍以上のファイルサイズ設定や Direct I/O の活用を標準とする

IO500 等の国際基準に準拠しつつも、「Stonewall time」等の適切な実行時間を設定することで、一過性の性能ではなく、持続可能なスループットを測定する

各ツールの概要について以下に示す。Athena++、R2D2およびNICAMについては「次世代計算基盤に係る調査研究事業　委託業務成果報告書（令和6年度）【システム研究調査（理化学研究所）チーム 】令和6年度成果報告書（2）」を参照。

- IOR

  IOR は様々な I/O インターフェースとアクセスパターンを用いてファイルシステムの I/O 性能を評価できるベンチマークツールである。I/O インターフェースには HDF5、HDFS、S3、NCMPI（NetCDF を並列アクセスするための API）、MMAP または RAODS が使用できる。また、アクセスパターンとして各 MPI プロセスが同じファイルへの I/O を行う Single Shared File および各 MPI プロセスが一意のファイルへ I/O を行う File Per Process が使用できる。また、IOR のバージョン 4.0 から GPUDirect Storage を用いた GPU の I/O 性能を評価も可能である。そのため、バンド幅の定量的な評価として使用可能である。

- mdtest

  mdtest は共有ファイルシステムに対してファイルシステムのメタデータ性能を評価できるベンチマークツールである。ファイルの作成、情報取得、および削除操作を行う。各プロセスは単一ディレクトリに対して各操作を実施するか、各プロセスが一意のディレクトリ配下に各操作を実施するかを指定できる。そのため、IOPS の定量的な評価として使用可能である。

- IO500

  IO500 は HPC（High-Performance Computing）システムのストレージ性能を評価するために作成されたベンチマークツールである。実際のアプリケーションで発生する様々なアクセスパターンを標準化した検証方法により評価する。これにより異なるシステム間でのストレージ性能を比較することができる。

  IO500 で使用されるワークロードを表 3-28 に示す。IO500 の各項目は 300 秒間実行した結果を評価に使用する。IOEasy、IOHard の API は IOR と同じ API を指定可能である。

  IO500 はストレージ要件の性能とは目的が異なるベンチマークツールである。そのためストレージ要件を評価するベンチマークツールには適していない。しかし、安定性の評価としては使用可能である。

表 3-28　IO500 の測定パターン詳細

項目	測定内容	ツール名	ファイルアクセスパターン
IOEasy	IO スループット性能	IOR	・FPP による read/write ・任意の IO サイズを指定可能
IOHard	IO スループット性能	IOR	・SSF による read/write ・各プロセスが 47008 バイトのデータを一定間隔(47008 バイト×プロセス数)で繰り返し

			read/write
MDEasy	メタデータ性能	mdtest	・各プロセスがそれぞれ個別のディレクトリ配下に 0 バイトの空ファイルを作成/stat/削除
MDHard	メタデータ性能	mdtest	・全プロセスが単一ディレクトリ配下にファイルを作成/write/stat/read/削除(3901 バイトを write/read)
Find	ファイル探索性能	pfind	・指定ディレクトリ配下でパターンに一致するファイルを検索(MDHard で作成したファイルを対象)

- GDSIO

  NVIDIA の GPUDirect Storage 向けの I/O ベンチマークツール。単一プロセスを 1 つの GPU に割り当て、GPU メモリとストレージ間の直接データ転送パスのスループットとレイテンシの測定ができる。I/O インターフェースやアクセスパターンを指定するオプションはない。

  単一ノード上で実行することを想定したツールのため、複数ノードや複数プロセスを用いた評価には向いていない。

- elbencho

  elbencho はファイル、オブジェクト（S3）またはブロックストレージのレイテンシ、スループット、IOPS 性能を評価できるベンチマークツールである。GPUDirect Storage に対応しており、GPU によるデータ転送性能を直接評価することが可能である。MPI に非対応であり、様々な環境で実行できる。I/O インターフェースは POSIX のみとなっている。アクセスパターンとして Single Shared File および File Per Process が使用できる。

  バンド幅、IOPS の定量的な評価に使用可能である。しかし、MPI に非対応であるため、第 2 階層の要件である MPI-IO による Single Shared File 方式による書き出しの性能を評価することはできない。

- Deep Learning I/O Benchmark

  Deep Learning のアプリケーションの I/O パターンと動作をエミュレートすることを目的としたベンチマークスイート。AI 性能評価指標である MLPerf Storage Benchmark の基盤技術として採用されている。様々なデータ形式、データセット構成に対応し、実際の Deep Learning アプリケーションと同様の設定パラメータを使用する。幅広い Deep Learning アプリケーションの I/O パターンをエミュレートする。

  本ツールは評価する機械学習のワークロードによって評価方法が変わるため定量的なファイルシステム性能の評価に用いるかは詳細設計以降で検討が必要である。

  Deep Learning I/O Benchmark の入力データ、アクセスパターンなどの詳細は以下の通り。

  - 入力データ（学習データ）
    - TFRecord、HDF5、NPZ、PNG、JPEG、CSV など
  - アクセスパターン
    - Read：学習データ
    - Write：チェックポイント（エポックごとに重みなどのダミーデータを保存）
    - PyTorch/TensorFlow の API を利用
  - モデル（I/O パターンエミュレート用）

- U-Met3D（画像セグメンテーション）、ResNet-50（画像分類）、BERT/LlaMA（自然言語処理）、CosmoFlow（宇宙論パラメーター推定）など

- ベンチマーク出力
  - サマリーレポート：スループット、レイテンシ、総 I/O 回数、バイト数、データ読み込み量、出力時間など
  - プロファイリング：各ファイル I/O 操作の開始、終了時間、read/write バイト数、ファイルパス

3.4.4.2.1 第 1 階層

第1階層のバンド幅およびIOPSを測定するためのベンチマークおよび確認方法について記載する。ベンチマークのオプションは一部記載するが、詳細については計算ノードの搭載メモリサイズが決定後に検討とする。

第1階層のファイルシステムをローカルファイルシステムまたは共有ファイルシステムとするかは検討中である。本章で述べているツール類は共有ファイルシステムを想定している。そのため、ローカルファイルシステムの場合はFile Per Processやプロセスごとの個別ディレクトリに対して評価を行う必要があり、評価に使用するローカルファイルシステムのディレクトリ構成は各計算ノードで同じにする必要がある。

共有ファイルシステムの場合はSingle Shared Fileや同一ディレクトリに対して評価を行う。

- バンド幅：計算ノードの全メモリの内容を 2 分以下で書き出すことができること。読み込みも同等以上の性能であること。
  - ローカルファイルシステム(例)
    # IOR -a HDF5 -i 5 -F -w -r -t 1m -b <memory_size ÷1m> -o <outputfile>
    -F オプションが File Per Process の指定である。指定がない場合は Single Shared File となる
  - 共有ファイルシステム(例)
    # IOR -a HDF5 -i 5 -w -r -t 1m -b <memory_size ÷1m> -o <outputfile>
  - 確認方法
    IOR 出力の write と read 時間がそれぞれ 2 分以下

- IOPS：単一計算ノードあたりメタデータ処理（ファイル生成・削除、Read/Write を含む）IOPS が 8K 以上であること
  - ローカルファイルシステム(例)
    # mdtest -F -i 5 -n 1000 -w 1024 -e 1024 -r -u -d <outputdir>
    -u オプションがプロセスごとの個別ディレクトリの指定である。指定がない場合は同一ディレクトリとなる。
  - 共有ファイルシステム(例)
    # mdtest -F -i 5 -n 1000 -w 1024 -e 1024 -r -d <outputdir>
  - 確認方法
    ファイル作成、write、read、削除の出力値（operation/sec）÷ノード数の値が 8K 以上

3.4.4.2.2 第 2 階層

第2階層のバンド幅およびIOPSを測定するためのベンチマークおよび確認方法について記載する。ベンチマークのオプションは一部記載するが、詳細については計算ノードの搭載メモリサイズが決定後に別途検討が必要である。また、第2階層のファイルシステムのバンド幅の評価ではMPI-IO による Single Shared File 方式による評価となっている。Single Shared File 方式のアクセスパターンとして図 3-45に示す2パターンが考えられる。図中のセグメントは各計算ノード上のプロセスが書き込みを行うデータ量を示す。（A）のアクセスパターンは計算ノードのメモリサ

イズ分のデータを1つのセグメントとし書き込みを行う。これを全計算ノード分の書き込みを繰り返す。（B）のアクセスパターンは計算ノードのメモリサイズ分のデータを複数セグメントに分割する。分割したセグメントの 1 つの書き込みを全計算ノード分繰り返し行う。これを分割したすべてのセグメントの書き込みが終わるまで繰り返す。

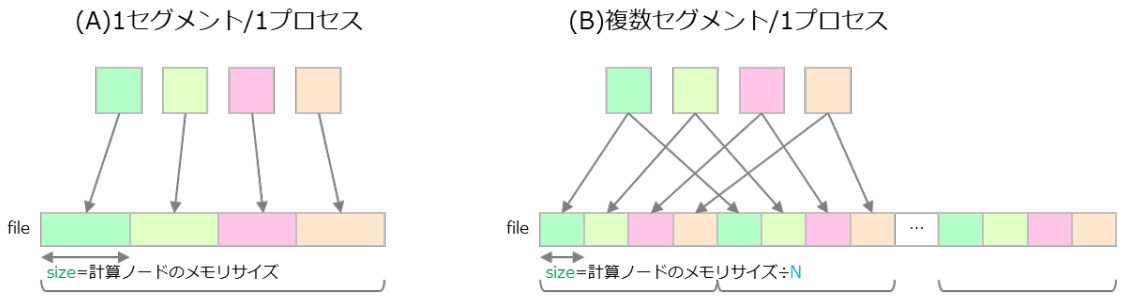


図 3-45 Single Shared File のアクセスパターン

上記のアクセスパターンについてはIORの-sオプションの指定で実現できる。-sオプションはセグメント数を指定するオプションであり、デフォルト値は1である。（A）、（B）それぞれのI/Oパターンのオプション例を以下に示す。

（A）IOR -w -r -t size -b size -s 1 -o /path/to/file

（B）IOR -w -r -t size -b （size/N） -s N -o /path/to/file

- バンド幅：計算ノードの全メモリの内容を MPI-IO による Single Shared File 方式によって 10 分以下で書き出すことができること。読み込みも同等以上の性能であること。
  - 共有ファイルシステム
    # IOR -a MPIIO -c -i 5 -w -r -t 1m -b <memory_size ÷ 1m> -s 1 -o <outputfile>
  - 確認方法
    IOR 出力の write と read 時間がそれぞれ 10 分以下

- IOPS：全ノードから同時にリクエストされる I/O 処理に対して、I/O 処理が停止することなく安定して稼働し 1 ノードあたり IOPS が 1K 以上の性能であること。
  - 共有ファイルシステム
    # mdtest -F -i 5 -n 1000 -w 1024 -e 1024 -r -d <outputdir>
  - 確認方法
    ファイル作成、write、read、削除の出力値（operation/sec）÷ノード数の値が 1K 以上

3.4.5 複数予算想定下の構成検討とコスト推定

本章では、ストレージベンダーに対して実施した技術調査の内容、および技術要素の選択肢について、複数予算への対応を踏まえた検討内容を述べる。

3.4.5.1 期待される仕様

本節では、第1階層・第2階層ストレージに期待される仕様のうち、特にフラッシュデバイス仕様および第2階層ストレージの分割・設置構成について、ストレージベンダーに対して実施した技術調査のヒアリング項目をベースに述べる。

3.4.5.1.1 デバイスの種類

本項では、第1階層ストレージを構成するフラッシュデバイス（SSD）について、コスト推定および設計成立性の観点から期待される仕様の検討内容を記載する。

1. 基本情報

SSDの世代・接続方式・形状は、性能上限、実装密度、保守性に影響するため、基本情報として整理する。

- 世代（PCIe Gen）
  - 性能目標に照らし、インターフェース世代が要件に整合することが求められる。ノードアーキテクチャと整合を取るため、搭載可能な世代・レーン構成であることが求められる。例として Gen5 / Gen6 / Gen7 が考えられる。
- メーカー名
  - 調達性と長期運用を成立させるため、供給継続性および FW 更新・保守サポートが担保されることが求められる。
- 製品グレード（Grade）
  - 稼働温度・保証時間・耐久保証などが異なるため、用途に応じたグレード選択が必要である。例として Enterprise / Datacenter / Client が考えられる。
- 製品名/コントローラー名
  - 世代・搭載機能・実績を把握するため、製品名およびコントローラー情報が明確であることが求められる。
- Interface
  - 物理接続および搭載方式を成立させるため、スロット形状と適合するインターフェースであることが求められる。例として PCIe x4, x8 / U.2 / U.3 / E3.S / E3.L などが考えられる。
- プロトコル（Protocol）
  - OS/ドライバ互換性および管理機能を確保するため、対応プロトコルが要件を満たすことが求められる。例として NVMe / SATA / SAS が考えられる。
- フォームファクタ（Form Factor）
  - ノードアーキテクチャおよび実装密度・保守性（ホットスワップ性）を成立させるフォームファクタの選択が必要である。例として E1.S / E3.S / E3.L / U.2 / U.3 / M.2 が考えられる。

2. NAND構成情報

NANDの構造は性能・耐久・コストの前提条件となるため、構成情報として整理する。

- Level Cell

- 容量・性能・耐久性・コストのバランスを確保するため、セル種別を精査する。例として SLC / MLC / TLC / QLC が考えられる。
- NAND チップ技術
  - 容量・性能・耐久性に影響するため、層数およびセル構造が明確であることが求められる。例として 3D NAND 層数（96L～232L）/構造（CTF / FG） が挙げられる。
- コントローラーチャネル構成
  - 内部並列度と性能上限に影響するため、チャネル数が要件を満たすことが求められる。例として 8ch～16ch が考えられる。
- ページサイズ
  - 書込み効率および GC（Garbage Collection）挙動に影響するため、ページサイズが明確であることが求められる。例として 16KB～32KB が考えられる。

3. 容量・コントローラー層

容量とコントローラーは台数設計および単価・消費電力に直結するため、仕様を整理する。

- 容量
  - サーバ台数、ラック数、総重量、コストを見積もるため、SSD1 台あたりの容量を選定する必要がある。例として 7.68TB / 15.36TB / 30.72TB が考えられる。
- コントローラーIC
  - 性能・安定性に影響するため、対応機能および実績が明確であることが求められる。

4. 性能情報

期待性能（帯域・IOPS・遅延）を満たすため、代表指標を整理する。

- レイテンシ（遅延時間）
  - 小 I/O の応答性を評価するため、遅延時間を把握する。例として Read 80µs / Write 15µs が考えられる。
- Sequential Read/Write
  - 大容量転送時の上限性能を見積もるため、シーケンシャル性能を把握する。例として Read 14,000 MB/s / Write 7,000 MB/s が挙げられる。
- Random Read/Write IOPS（4KB）
  - 小ブロック I/O を評価するため、4KB ランダム IOPS を把握する。例として Random Read 5,500K IOPS / Random Write 900K IOPS が考えられる。

5. 信頼性・耐久性（Quality）

長期運用を成立させるため、耐久と故障率指標を整理する。

- RND_TBW（ランダム書込耐性）/ SEQ_TBW（シーケンシャル書込耐性）
  - 安定動作の継続性を評価するため、書込み耐性を把握する。例として RND 10 PBW / SEQ 30 PBW が考えられる。
- AFR（年間故障率）/ MTBF（平均故障間隔）
  - 故障見積・保守計画の前提として、故障率指標の把握が求められる。例として AFR 0.35% / MTBF 2,000,000h が挙げられる。

6. 消費電力・熱設計

ラック電力および冷却設計に直結するため、電力・熱関連仕様を整理する。

- 動作時最大消費電力
  - 施設の受電計画および運用費見積に影響するため、最大消費電力を把握する。例として 25W（E3.S）/ 40W（E3.L） が考えられる。
- 冷却設計・熱対応
  - 安定動作を継続するため、必要な冷却機構および熱対策を備えることが求められる。例として、エアフロー設計、ヒートシンク、サーマルスロットリング対応が考えられる。

7. 物理仕様

実装制約および搭載密度の観点から、寸法・重量を整理する。

- 外形寸法（W×H×D）
  - 物理干渉やトレイ適合確認に用いる。例として 76 × 111 × 7.5 mm が挙げられる。
- 重量
  - ラック荷重や保守作業性の観点から確認する。例として 120 g が考えられる。

8. 発売・価格情報

コスト推定の算出根拠とするため、価格情報を整理する。

- 発売時期 / 発売予測時期
  - 導入計画・調達リードタイムに影響する。例として 2028 Q1 サンプル出荷予定、2029 Q1 量産予定が考えられる。
- 参考価格
  - コスト推定の算出根拠として価格情報を確認する。

9. 特記事項

要件との整合や注意点を確実に引き継ぐため、補足情報を記載する。例として、容量を考慮するとE3.S、冷却を考慮するとEx.x等の判断材料を記載することが考えられる。

3.4.5.1.2 分割・設置構成

本項では、第2階層ストレージの段階導入および複数年度予算への対応に向けて、期待される分割・設置構成に関する検討内容を記載する。

1. 使用デバイス種類

要求性能とコスト効率の両立を図るため、媒体構成の選択が必要である。

- ALL フラッシュ構成
  - 高性能領域（メタデータ・高 IOPS 用途）での I/O 応答性を確保できる構成が求められる。例として、NVMe、E3.S、Gen5-6 対応が考えられる。
  - 性能要件を満たしつつ実装・運用を成立させる構成が可能であることが求められる。例として、全 NVMe 構成、ZFS ベース、RAID-10 / Erasure Coding（EC）構成が考えられる。
- フラッシュ＋HDD 構成（ハイブリッド）
  - 大容量・コスト効率重視領域に適用できることが求められる。例として、高頻度アクセスデータをフラッシュに配置する構成が考えられる。

- 性能と容量の両立を図るため、キャッシュ階層を適切に設計できることが求められる。例として NVMe＋SAS HDD＋HSM 構成 が挙げられる。

2. 容量・性能要件

コスト推定の前提条件を明確化するため、総容量・帯域・IOPSの想定値を整理する。

- ALL フラッシュ
  - 台数設計とコスト推定に反映するため、総容量を定量化する。例として、PB 級構成の可否が考えられる。
  - 性能の目標値を設定するため、帯域を定量化する。例として、GB/s～TB/s クラスの性能が考えられる。
  - 小ブロック I/O 性能を評価するため、IOPS（4KB read/write）を定量化する。
- フラッシュ＋HDD
  - キャッシュ階層性能（burst 帯域、write buffer 性能）、HDD 層順次性能（HDD 単体の順次 R/W 性能）、混在 I/O 性能（変動率） の観点で整理する。

3. 冗長化方式

障害時にも運用継続を可能とするため、冗長化方式を整理する。

- ストレージサーバー
  - 障害発生時も運用を継続し、運用管理を簡素化するため、適切なモード構成であることが求められる。例として、Active/Active または Active/Standby が考えられる。切り替え時間（フェイルオーバー時間）も確認する。
  - コントローラー障害時のデータ保護を確保するため、同期方式を備えることが求められる。例として、NVRAM ミラーや ZIL ログ等の同期方式が考えられる。
  - 安定した I/O を継続するため、要件に適合した通信方式であることが求められる。例として、InfiniBand、RoCE、Ethernet が考えられる。
- シェルフ
  - シェルフ単位の可用性を確保するため、構成冗長を備えることが求められる。例として、デュアルパスおよび電源冗長が考えられる。合わせて、ケーブル冗長の有無も確認する。
  - I/O 中の経路断を回避するため、接続冗長を備えることが求められる。例として、SAS スイッチ冗長または NVMe-oF 経路冗長が考えられる。
  - 運用停止を回避するため、冗長かつ交換容易であることが求められる。例として、冗長 FAN / 冗長電源、およびホットスワップ対応が考えられる。
- DISK
  - ディスク故障時のデータ損失を防ぐため、データ保護方式を備えることが求められる。例として、RAID6、RAID10、Erasure Coding（EC）が考えられる。合わせて、再構築（リビルド）時間も確認する。
  - 障害時の復旧性を確保するため、メタデータにも冗長性を備えることが求められる。例として、MDS/MDT の HA 構成が考えられる。
  - 障害・再起動時にも整合性を維持するため、一貫性保証を備えることが求められる。例として、journaling または write ordering が考えられる。

4. 拡張性・段階導入

階導入および複数年度予算への対応を可能とするため、拡張単位と構成遷移を明確化する必要がある。

- 分割単位

- 段階導入を成立させるため、拡張方式が分割可能であることが求められる。例として、ノード単位またはシェルフ単位の拡張が考えられる。スケールアウト可否も確認する。
- 複数年度予算に合わせた増設を可能とするため、容量追加単位が明確であることが求められる。例として、シェルフ単位またはラック単位の増設が考えられる。柔軟な段階導入の可否も確認する。

- 構成図
  - 初期導入の成立性を確認するため、現行構成（初期導入構成）を提示できることが求められる。例として、ベンダ提示の構成図を入手することが考えられる。
  - 将来拡張の成立性を確認するため、拡張フェーズ別の将来拡張構成を提示できることが求められる。例として、フェーズ別構成図およびスケールアップ時の見積可否が考えられる。
  - 設計自由度を確保するため、リソースマッピングが柔軟であることが求められる。例として、MDS / OSS / Client の配置方針を整理することが考えられる。

5. 消費電力

電源設備計画および運用費の見積に直結するため、電力見積の単位を定義する必要がある。

- 電力構成
  - 受電容量との整合を取るため、システム全体稼働時の最大消費電力を把握できることが求められる。kW 単位での提示を想定する。
  - 分電盤等の計画に必要な粒度を確保するため、分割最小単位の消費電力を把握できることが求められる。例として、最小ノード単位またはシェルフ単位が考えられる。
  - 段階導入の成立性を電力面から評価するため、拡張時の追加電力量を把握できることが求められる。例として、スケールアウト時の追加電力が考えられる。
  - PDU 等の設計に反映するため、ラック単位の消費電力を把握できることが求められる。例として、ラック全体の総電力（PDU 単位）が考えられる。

6. UPS の必要性

電源断時の保護方針に応じた設計を行うため、UPSの要否と容量を整理する。

- UPS 構成
  - 電源断時の保護要件を満たすため、UPS の有無と容量を定義できることが求められる。例として、保持時間および給電経路の設計が考えられる。合わせて、短時間停電対応および UPS レス構成の可否も確認する。

7. ラック

導入コストと可搬性に影響するため、設置要件（規格・台数・重量・搭載配置）を明確化する。

- 規格
  - ラック搭載・配置設計を成立させるため、ラック規格を明確化できることが求められる。例として、EIA / JIS / Open Compute 等が考えられる。
  - 冗長構成を含めた設置計画を立てるため、必要台数を明確化できることが求められる。例として、構成フェーズごとの台数提示が考えられる。
  - 設置スペースと搬入条件を満たすため、ラックサイズ（W×D×H）を明確化できることが求められる。例として、600×1,000×42U が考えられる。
  - 耐荷重条件を満たすため、搭載時総重量を明確化できることが求められる。例として、ストレージ搭載時のラック重量（総重量）が考えられる。

- 実装・保守・冷却設計を成立させるため、搭載構成図を提示できることが求められる。例として、サーバ/シェルフの搭載数・位置・空冷経路を可視化した図が考えられる。
- 配線・構成管理を成立させるため、ストレージ構成要素（サーバー数・シェルフ数・配線構成）を明確化できることが求められる。例として、ベンダ提供図面の入手が考えられる。

8. 予算の内訳

複数年度での導入計画を成立させるため、初期費用に加えて運用費を含む内訳の整理が必要である。

- コスト構成
  - 全体計画の総額を把握するため、最大構成時のコストを算出できることが求められる。例として、初期導入費＋年間保守＋電力コストが考えられる。
  - 段階導入シナリオを検討するため、分割最小単位のコストを把握できることが求められる。例として、ノード単位またはシェルフ単位の価格内訳が考えられる。
  - 拡張予算の見積精度を確保するため、増設単位の追加コストを把握できることが求められる。

9. 冷却・環境

設置環境の制約に適合させるため、冷却方式と効率を整理する必要がある。

- 冷却方式
  - 施設側の設備条件に適合させるため、対応可能な冷却方式であることが求められる。例として、空冷/水冷/温水冷却が考えられる。合わせて、温水冷却・液浸冷却の工夫有無も確認する。
  - 高密度構成での熱設計を成立させるため、高い冷却効率を確保できることが求められる。例として、吸排気構造およびラック配置制約が考えられる。

10. 価格・提案情報

採否判断および総コストに影響するため、提案の差別化要素を明確化する。

- 提案特徴
  - 付加価値による導入効果を期待するため、独自技術・付加価値が提示されることが望ましい。例として、温水冷却技術/独自 EC/QoS 制御が考えられる。

3.4.5.2 構成検討

本節では、前節で述べたストレージベンダーへのヒアリング項目の中から、特にコスト推定に影響する内容を抽出し、第1階層ストレージおよび第2階層ストレージの構成要素ごとに、想定されるスペックの選択肢を示す。なお、本節で提示する選択肢や推定値は、将来の技術動向によって変わる可能性がある。本検討結果と、各ストレージベンダーから提供された情報を踏まえ、詳細設計においても将来動向の反映を含め検討を継続し、最終的な構成を決定する。

3.4.5.2.1 使用するデバイスの種類

第1階層ストレージで使用するデバイスは、2030年のシステム稼働開始を念頭に、安定供給が可能な最新技術に基づいたSSDを選定することを基本方針とする。将来的なストレージ技術の動向およびベンダ情報を踏まえ、現時点で想定されるSSDの主要スペックの選択肢を以下に示す。なお、特にSSDの容量とLevel Cellについてはアーキテクチャ担当と連携し、詳細設計にて検討する。

- 基本情報（試算の前提となる基本仕様）
  - PCIe 世代：Gen5, Gen6, Gen7
    （ファイルシステムの性能効率や SSD の容量効率に依るが、性能目標の達成には Gen7 が必要となる見込み。今後ストレージベンダーの情報提供やベンチマークにより精査し、Gen 6/7 の混載や異なる世代の段階導入も選択肢に検討）
  - 製品グレード：エンタープライズ、データセンター（24 時間 365 日稼働時に高い信頼性、耐久性、性能を確保できるエンタープライズグレードが有力。分類はベンダ依存）
  - インターフェース：PCIe GenX x4, x8（PCIe 世代は前述の通り。4 レーンが主流）
  - プロトコル：NVMe, SAS（性能を最大限に引き出すのは NVMe）
  - フォームファクタ：E3.S, E3.L, U.2, U.3（EDSFF は高密度・高性能な実装が可能）
  - 容量： 1 ノードあたり SSD 4～8 台（搭載メモリー量の 3 倍以上が必要）
  - 重量：製品依存。E3.S/L の場合、100～150g。
- NAND 構成
  - Level Cell：SLC, MLC, TLC, QLC（容量、速度、信頼性、コストのバランスを考慮）
  - NAND チップ技術：3D NAND
- 性能 (NDA 情報を含むため省略)
- 信頼性・耐久性 (NDA 情報を含むため省略)
- 消費電力・熱設計
  - 消費電力：(NDA 情報を含むため省略)
  - 冷却機構：空冷/水冷、ヒートシンク形状/エアフロー対策/サーマルスロットリング対応

第2階層ストレージは、大容量・高性能・高コスト効率を実現するため、Flash+HDDおよびAll Flashの両方を候補として検討した。第2階層ストレージで使用するデバイスは、2030年のシステム稼働開始を念頭に、安定供給が可能な最新技術に基づいたSSD/HDDを選定することを基本方針とする。将来的なストレージ技術の動向およびベンダ情報を踏まえ、現時点で想定されるSSDおよびHDDの主要スペックの選択肢を以下に示す。

- 基本情報（試算の前提となる基本仕様）
  - PCIe 世代(SSD)：Gen5, Gen6, Gen7（安定供給可能な最新世代は Gen6 の見込み）
  - 製品グレード：エンタープライズ、データセンター（24 時間 365 日稼働時に高い信頼性、耐久性、性能を確保できるエンタープライズグレードが有力。分類はベンダ依存）
  - インターフェース：
    - SSD：PCIe GenX x4, x8（PCIe 世代は前述の通り。4 レーンが主流）
    - HDD：SAS, NL-SAS
  - プロトコル：
    - SSD：NVMe, SAS（性能を最大限に引き出すのは NVMe）
    - HDD：SAS, NL-SAS
  - フォームファクタ：
    - SSD：E3.S, E3.L, U.2, U.3
    - HDD：2.5 インチ、3.5 インチ
  - 容量：(NDA 情報を含むため省略)
  - 重量：

        - ✧ SSD：製品依存。2.5 インチフォームファクターで 80～150g
        - ✧ HDD：製品依存。3.5 インチで 650～750g
    - NAND 構成（SSD）
        - ➢ Level Cell：SLC, MLC, TLC, QLC（容量、速度、信頼性、コストのバランスを考慮）
        - ➢ NAND チップ技術：3D NAND
    - 性能
        - ➢ 前述のスペックの範囲では、以下の性能を達成可能と推定（今後 2 年間の技術的ブレイクスルーは未考慮）。
            - ✧ SSD(1 台あたり)：(NDA 情報を含むため省略)
            - ✧ HDD(1 台あたり)：(NDA 情報を含むため省略)
    - 信頼性・耐久性
        - ➢ 前述のスペックの範囲では、以下の性能を達成可能と推定。
            - ✧ SSD：(NDA 情報を含むため省略)
            - ✧ HDD：(NDA 情報を含むため省略)
    - 消費電力・熱設計
        - ➢ 消費電力：(NDA 情報を含むため省略)
        - ➢ 冷却機構：空冷／水冷、水冷冷却ドア（機器は空冷）
    - ネットワークデバイス構成
        - ➢ Scale-out ネットワーク（計算ネットワーク）
            - ✧ 接続規格：Ultra Ethernet または InfiniBand
            - ✧ 数量：ストレージサーバー（meta/data）1 対あたり 4～8 本
            - ✧ 備考：ストレージネットワークと計算ネットワークは共有とし、通信経路分散の仕組みを用意することを仮定して試算。ネットワーク分離はコスト次第であり、詳細設計で引き続き検討。
        - ➢ 管理用ネットワーク
            - ✧ 接続規格：1GbE または 10GbE
            - ✧ 数量：ストレージサーバー（meta/data）1 対あたり 6～14
            - ✧ 備考：システム全体の管理ネットワークのうち、ストレージ割り当て分

3.4.5.2.2　容量、バンド幅、IOPS

第1階層ストレージの容量、バンド幅、IOPSを推定する。前述のデバイス情報を前提として、第1階層ストレージの容量、バンド幅、IOPSを推定する。搭載メモリ量を5TB、計算ノード台数を4,000と仮定した場合の推定値、目標値は以下の通り。なお、I/O性能の推定値には、今後2年間の技術的ブレイクスルーは考慮されていない。

(詳細についてはNDA情報を含むため省略)

第2階層ストレージの容量、バンド幅、IOPSを推定する。前述のデバイス情報、後述の冗長化方式、拡張性を前提として、第2階層ストレージの容量、バンド幅、IOPSを推定する。搭載メモリ量を5TB、計算ノード台数を4,000と仮定した場合の要求仕様と比較して試算した。

- All Flash の場合: (NDA 情報を含むため省略)
- Flash+HDD の場合: (NDA 情報を含むため省略)

容量・性能目標は全体予算中のストレージ配分額を前提とするため、当該配分額の変動により目標値は制約

を受ける。例えば、配分額が当初計画比50%となった場合、容量・性能の目標値も概ね当初比50%に制限される。

3.4.5.2.3　冗長化方式

第1階層ストレージの冗長化方式として現時点で想定される選択肢を記載する。

- 計算ノード全体：ノードローカルストレージ、ニアノードローカルストレージ
  - 詳細は 3.4.2.3.1 項を参照。現時点では、個々の計算ノードが持つローカルストレージをジョブ内で相互利用する構成を仮定している。
- データ保護方式：RAID (RAID6, RAID10), Erasure Coding, ミラーリング
  - 各ノードのローカルストレージが高性能 SSD 4～8 台で構成され、ノード内のデータ保護方式としては RAID6、RAID10 の場合、物理デバイスの合計容量に対し実際に利用可能な容量は 7 割程度となる。
  - 選定するファイルシステムによっては、Erasure Coding によって複数ノード間でのデータ保護を実装可能。

第2階層ストレージの冗長化方式として現時点で想定される選択肢を記載する。

- ストレージサーバー：Active/Active, Active/Standby, 複数ノード Active
  - データの運用継続性（高可用性）と性能向上の観点から、ストレージサーバーは Active/Active の高可用性(HA)構成を仮定。
- 物理筐体：コントローラー冗長、ネットワークパス二重化、冗長電源、エンクロージャ多重接続
  - ストレージサーバーを格納するコントローラーやシェルフについても、筐体レベルでの冗長化（ネットワークパスの二重化、冗長電源、エンクロージャの多重接続）を仮定。
- データ保護方式：RAID6（相当）, Erasure Coding, ミラーリング
  - SSD/HDD で構成されるストレージプールのデータ保護方式。RAID6 相当または Erasure Coding 等の冗長化を行う場合、物理デバイスの合計容量に対し実際に利用可能な容量は 7 割程度となる。

3.4.5.2.4　拡張性・段階導入の可否

第1階層ストレージの拡張性・段階導入の可否について、現時点で想定される選択肢を記載する。

- 前提
  - 第 1 階層ストレージは、各計算ノードのローカルストレージ（複数 SSD で構成）を活用し、ノード間ネットワーク経由で連携する 1 つの分散ファイルシステムであると想定
- 導入段階：計算ノード構築に応じて順次/将来のノード数拡張に備え
- 分割単位（拡張方式）：ノード単位、ノード群単位
  - ストレージ専用サーバが必要な場合はノード群単位の拡張も想定される。
- 容量追加単位：ドライブ（SSD）単位、ノード単位、ノード群単位
- ファイルシステム統合：（各段階で）同一、異なる
  - 基本的には、拡張するドライブやノード群は既設の第 1 階層ファイルシステムに統合されることを想定
  - 第 1 階層ファイルシステムの選定次第では、統合が不要な可能性あり（ジョブ実行時に scrach 領域を動的に作成）
- 拡張性（上限）：計算ノード数 5,000～10,000、ノードあたり容量 128GB～50TB
- 拡張時の運用影響：運用停止あり / 瞬断あり / なし

- ファイルシステムの拡張時における第 1 階層ストレージの運用停止の要否

第2階層ストレージの拡張性・段階導入の可否について、現時点で想定される選択肢を記載する。

- 予算措置：初期導入時一括、複数年度予算（機器調達費含む）
- 導入段階数：1～6
  - 段階数が少ない場合、段階ごとの容量・性能差は大、各年度の必要予算は大、導入工数や運用影響は小。
  - 段階数が多い場合、段階ごとの容量・性能差は小、各年度の必要予算は小、導入工数や運用影響は大。
- 分割単位（拡張方式）：ノード、シェルフ、ラック
  - ノード単位、シェルフ単位の拡張に対応していても、設計・運用上の合理性からラック単位を選択することは考えられる。
  - ドライブの寿命のみならずコントローラー等を含めた筐体全体のライフサイクル管理の観点を考慮する。
  - 段階導入の進展に伴い退役させる機器を第 2 階層ストレージシステムから分離、除外可能である必要がある。
- 容量追加単位：ドライブ（プール）、シェルフ、ラック
  - ドライブ（プール）単位の拡張を行う場合、コントローラーやラックは予め設置しておくため、コントローラーの寿命を考慮する必要がある。
- ファイルシステム統合：（各段階で）同一、異なる
  - 各段階で 1 つのファイルシステムを構成する（計算ノードからは複数ファイルシステムを並行してマウントする）か、設置済みのファイルシステムと統合するか（あるいはその両方）を選択する。
  - 各段階において異なるファイルシステム種別を選択し、用途に応じて使い分けることは可能。一体運用により性能・容量・利便性のメリットを享受できるかを考慮する。
- 全体容量（最小～最大）：
  - 段階導入による既存設備残置等により、合計容量が一時的に 0.5～3 倍の範囲で変動し得る。運用開始時点および最大時点で達成する容量は、詳細設計以降で検討する。
  - 現時点では、最大の合計容量を 6 分割して導入すると仮定して試算している。
- 拡張時の運用影響：運用停止あり / 瞬断あり / なし
  - ファイルシステムの拡張時における第 2 階層ストレージの運用停止の要否

3.4.5.2.5　消費電力

第2階層ストレージの消費電力は、3.4.5.2.7項のラック数・サーバ数を基に推計する。試算結果を以下に示す。

- 総消費電力(第 2 階層全体)：(NDA 情報を含むため省略)

3.4.5.2.6　UPS の必要性

UPSの必要性について、以下の通り検討した。

- 停電発生時にストレージを安全に停止しデータを保護するため、「富岳 NEXT」においても「富岳」同様に UPS に相当する高度な電源供給の仕組みが求められる。「富岳」は施設側の高度な電源供給の仕組み（ガスタービン）により、UPS は不要である。
- 構築・設置・インテグレーション検討会に対し、ストレージ向けの UPS に相当する高度な電源供給の仕組みを必須要件としてインプットした。

- UPS の場合、停電発生後にストレージを安全に停止できる時間（20 分～30 分）を確保できる容量が必要。

3.4.5.2.7　ラック（規格、ラック数、ラックサイズ）やストレージサーバー数

本項では、前述のスペック（デバイス選定、容量・性能、冗長化、拡張性など）に基づき、第2階層ストレージのラック数やストレージサーバー数を試算した結果を述べる。ラック規格はEIA規格（19インチ42Uラック）を仮定した。本試算は現時点で入手可能な情報に基づいており、今後の技術動向やシステム仕様の検討結果に応じて継続的に見直すことを前提としている。

表 3-29 ストレージ機器数およびラック数(試算) (NDA 情報を含むため省略)

表 3-30 ネットワーク機器数およびラック数(試算) (NDA 情報を含むため省略)

図 3-46 ラック搭載図例（All Flash 構成：ストレージ本体）(NDA 情報を含むため省略)

図 3-47 ラック搭載図例（Flash+HDD 構成：ストレージ本体）(NDA 情報を含むため省略)

図 3-48 ラック搭載図例（共通：ネットワーク装置）(NDA 情報を含むため省略)

図 3-49 ラック配置図例（All Flash 構成）(NDA 情報を含むため省略)

図 3-50 ラック配置図例（Flash+HDD 構成）(NDA 情報を含むため省略)

3.4.5.2.8　予算の内訳

現時点では具体的な予算額の算出が困難なため、市場動向を注視し、ベンダ情報を継続して収集する。第2階層ストレージの予算は、3.4.5.2.7項に示した数量の内訳に基づき、詳細設計以降で継続して試算する。

3.4.6 詳細設計での検討事項

本章では、基本設計の検討結果を踏まえ、詳細設計において更なる精緻化や判断が求められる事項を整理する。これらは、基本設計段階で方向性を示したものの、より具体的な仕様決定や技術的検証が必要とされる領域である。また、将来的な運用や拡張を見据え、柔軟性を確保するために留意すべき観点も含めて提示する。

表 3-31 詳細設計での検討事項一覧

No	検討事項	該当章	備考
1	ノードローカルストレージとニアノードローカルストレージを詳細に比較検討し、第 1 階層ストレージのハードウェア構成を決定する。	3.4.2.3.1.1	
2	ストレージシステムに関する要件整理を行い、当該要件に見合うストレージシステムが導入されるよう、ストレージベンダーと協調しながらの設計を検討する。また、必要に応じて各ベンダから提供される情報をもとに性能見積を行い、提案されたストレージシステムについて妥当性の検証および比較検討ができるよう整理する。	3.4.2.3.3 3.4.2.4.4	
3	MPI-IO の非同期 I/O について、ファイルシステムでの対応要否を含めて検討を実施する。	3.4.2.3.3 3.4.2.4.4	
4	GPU による第 1 階層/第 2 階層ストレージの利用方法について検討する。	3.4.2.3.3 3.4.2.4.4	
5	各ベンダの情報をもとにファイルシステムの選定、および機能開発要否や複数のファイルシステムの組み合わせ要否の検討などを行っていく。	3.4.2.3.4 3.4.2.4.5	
6	ストレージサーバーの必要台数を検討する。	3.4.2.4.1.1	
7	第 2 階層ストレージの利用方法として、全体で 1 つのボリュームとして利用するのではなく、仮想的に分割して利用する構成も検討する。	3.4.2.4.1.1	
8	容量と性能のバランスから性能要件の書き込み性能を満たすことが難しい可能性があるため、要求性能の見直しを検討する。	3.4.2.4.1.1	
9	分割導入の具体的な解決策や導入方式の検討を実施する。	3.4.2.4.2	
10	第 1 階層/第 2 階層間のデータ転送について、「富岳 NEXT」が対応するジョブスケジューラやファイルシステムが決定した後に具体的な検討を行う。	3.4.3.1	
11	GPUDirect 利用観点を含めて AI 観点の第 1 階層/第 2 階層間のデータ転送パターンを検討する。	3.4.3.1.2	
12	アプリケーションと絡めてチェックポ [4]イント/リスタートの検討を行う。	3.4.3.1.2	
13	llio_transfer コマンドの以下の課題について検討する。 ・ユーザへの使い方の周知	3.4.3.1.2	

	・0 バイトのファイルの転送		
14	第 1 階層/第 2 階層間の各データ転送パターンのサポートについて、「富岳 NEXT」が対応するジョブスケジューラやファイルシステムが決定した後に具体的な検討を行う。	3.4.3.1.3	
15	対応する外部ストレージを決定する。	3.4.3.2	
16	外部ストレージと第 2 階層の連携方法およびデータ転送方法について、使用するソフトウェアやデータ転送サービスと合わせて比較/検討して決定する。	3.4.3.2	
17	富岳で課題とされている HPCI 共用ストレージなどの外部ストレージとの連携において、データの移動頻度や速度不足に起因した転送負荷の問題に対して具体的な対処方法を検討する。	3.4.3.2	
18	ストレージシステムの運用支援・性能分析ツールについて、選定したファイルシステムに不足する機能の開発項目を検討する。	3.4.3.3.1	
19	分割導入での各導入時の性能評価の仕方を検討する。	3.4.4.1	
20	ベンチマーク手法として複数回の実行回数やばらつきの範囲など安定性評価方法について検討する。	3.4.4.2	
21	ベンチマーク手法として GPU によるファイルシステム性能評価について検討する。	3.4.4.2	
22	ベンチマーク手法として他ジョブの外乱などの影響を含めた評価方法について検討する。	3.4.4.2	
23	ベンチマーク手法としてアプリケーションによるファイルシステム性能評価について検討する。	3.4.4.2	
24	ベンチマーク手法として elbencho や Deep Learning I/O Benchmark を用いた評価方法を検討する。	3.4.4.2	
25	ベンチマーク手法として IOR の IO サイズを計算ノードの搭載メモリ量決定後に検討する。	3.4.4.2.1	
26	段階導入における運用開始時点および既存設備残置時の合計容量の計画について検討する。	3.4.5.2.4	
27	ストレージ機器数・ラック数・予算の内訳について、ストレージベンダーから入手した情報を基に、検討を継続する。	3.4.5.2.8	
28	多量ファイルを N ノードに配布するツールの提供について、第 1 階層ファイルシステムの機能を踏まえて開発の要否を検討する。		
29	ストレージアーキテクチャについてストレージベンダーと継続的に議論し、ストレージ検討会の定例会にて共有する。		
30	第 1 階層ストレージの設計空間を更に絞り込む。特に SSD の容量・Level Cell、PCIe 世代、Scale-out ネットワークの分離要否を検討し、アーキテクチャ設計にインプットする。		
31	ストレージ装置を提供するベンダが採用するデバイス情報（特に SSD）を収集する。		
32	第 2 階層ストレージの制約事項（特にインターコネクト種別、消費電力）を検討し、アーキテクチャ設計・設備設計にインプットす		

	る。		
33	富岳 NEXT の予算で開発が必要な機能が発生した場合、理化学研究所様と連携して推進する。 （例：第 1 階層ファイルシステムにステージング機能がない、またはステージング機能とスケジューラーの連携が未サポートの場合）		
34	POSIX 互換性レベルと実効性能の最適化 DAOS 2.6 (DFuse) 等で見られる、既存アプリケーションを無変更で動作させるための高い POSIX 互換性と、I/O 高速化の両立について、詳細設計段階での実機検証による最終判断		
35	計算ノード側の物理リソース要件の確定 富岳における I/O 不安定性の主因となった「メモリ枯渇（接続情報、メタデータ保持等）」を回避するため、計算ノードへの SSD 搭載数や、メタデータ操作に必要なメモリ容量を詳細設計で厳格に確定させること		
36	ユーザに直接多様な転送ツール（Globus, Rclone, Aspera 等）を扱わせることによる運用の破綻を防ぐため、運用側で制御・品質担保を行うミドルウェアの具体的仕様を決定する 特に、0 バイトファイルの転送不全（llio_transfer の課題）の解消や、ジョブスケジューラとの API 連携機能を詳細設計に盛り込む		
37	ソフトウェアエコシステムの継続性と保守体制 運用開始後の継続的な改修・改善を担保するため、OSS コミュニティの活動状況や、ISV のサポートを選定の重要評価項目として引き継ぐ		

参照文献

Altair. (日付不明). Altair PBS Professional. 参照日: 2026 年 1 月 19 日, 参照先: https://altairjp.co.jp/pbs-professional

SchedMD. (2026 年 1 月 17 日). GitHub - SchedMD/slurm: Slurm: A Highly Scalable Workload Manager. 参照日: 2026 年 1 月 19 日, 参照先: https://github.com/SchedMD/slurm

OpenSFS and EOFS. (日付不明). Lustre. 参照日: 2026 年 1 月 19 日, 参照先: https://www.lustre.org/

Lawrence Livermore National Security, LLC. (2025 年 7 月 15 日). GitHub - hpc/mpifileutils: File utilities designed for scalability and performance. 参照日: 2026 年 1 月 19 日, 参照先: https://github.com/hpc/mpifileutils

HPCI. (日付不明). HPCI 共用ストレージ. 参照日: 2026 年 1 月 16 日, 参照先: https://www.hpci-office.jp/using_hpci/hardware_software_resource/2026/hpci_2026_st-1

Amazon Web Services, Inc. (2026). クラウドオブジェクトストレージ – Amazon S3. 参照日: 2026 年 1 月 16 日, 参照先: https://aws.amazon.com/jp/s3/

Google. (日付不明). Cloud Storage | Google Cloud. 参照日: 2026 年 1 月 16 日, 参照先: https://cloud.google.com/storage?hl=ja

Oracle. (2025 年 10 月 30 日). Object Storage | オラクル | Oracle 日本. 参照日: 2026 年 1 月 16 日, 参照先: https://www.oracle.com/jp/cloud/storage/object-storage/

Microsoft. (2026). Azure Blob Storage | Microsoft Azure. 参照日: 2026 年 1 月 16 日, 参照先: https://azure.microsoft.com/ja-jp/products/storage/blobs

Craig-WoodNick. (2026). Rclone. 参照日: 2026 年 1 月 19 日, 参照先: https://rclone.org/

The University of Chicago. (2026). Globus. 参照日: 2026 年 1 月 19 日, 参照先: https://www.globus.org/

Amazon Web Services, Inc. (2026). What is AWS DataSync? - AWS DataSync. 参照日: 2026 年 1 月 19 日, 参照先: https://docs.aws.amazon.com/datasync/latest/userguide/what-is-datasync.html

Google. (2026). Storage Transfer Service | Google Cloud. 参照日: 2026 年 1 月 19 日, 参照先: https://cloud.google.com/storage-transfer-service?hl=ja

Versity Software, Inc. (2026). ScoutAM - Large Scale Archival Storage - Versity. 参照日: 2026 年 1 月 19 日, 参照先: https://www.versity.com/products/scoutam/

HPSS Collaboration. (2026). Home - HPSS Collaboration. 参照日: 2026 年 1 月 19 日, 参照先: https://hpss-collaboration.org/

whamcloud. (2019 年 8 月 28 日). GitHub - whamcloud/lemur: Lustre HSM tools. 参照日: 2026 年 1 月 19 日, 参照先: https://github.com/whamcloud/lemur

Compute Canada. (2020 年 10 月 16 日). GitHub - ComputeCanada/lustre-obj-copytool: Object copytool for Lustre HSM. 参照日: 2026 年 1 月 19 日, 参照先: https://github.com/ComputeCanada/lustre-obj-copytool

CEA. (2024 年 11 月 22 日). Home · cea-hpc/robinhood Wiki · GitHub. 参照日: 2026 年 1 月 19 日, 参照先: https://github.com/cea-hpc/robinhood/wiki

Amazon Web Services, Inc. (2026). What is the AWS Command Line Interface? - AWS Command Line Interface. 参照日: 2026 年 1 月 19 日, 参照先: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html

s3tools.org. (2026). Amazon S3 Tools: Command Line S3 Client and S3 Backup for Windows, Linux: s3cmd, s3express. 参照日: 2026 年 1 月 19 日, 参照先: https://s3tools.org/s3cmd

VAST Data. (日付不明). Introduction to the VAST Management System. 参照日: 2026 年 1 月 7 日, 参照先: https://www.vastdata.com/resources/lab-guides/introduction-to-the-vast-management-system-lab-guide

XiLi. (2019 年 11 月 4 日). GitHub - Lustre Monitoring System. 参照日: 2026 年 1 月 7 日, 参照先: https://github.com/DDNStorage/LustrePerfMon

BrimJ.Michael. (2015 年 4 月 26 日). Monitoring Extreme-scale Lustre Toolkit. 参照日: 2026 年 1 月 7 日, 参照先: https://doi.org/10.48550/arXiv.1504.06836

Sivalingam, K., & Richardson, H. (2020, 10 20). Application IO Analysis with Lustre Monitoring Using LASSi for ARCHER. *High Performance Computing, 12321*(ISC High Performance 2020 International Workshops), 255-266. Retrieved from https://doi.org/10.1007/978-3-030-59851-8_16

HammondJohn. (2011 年 1 月 25 日). GitHub - lltop. 参照日: 2026 年 1 月 7 日, 参照先: https://github.com/jhammond/lltop

HammondJohn . (2012 年 4 月 25 日). GitHub - xltop. 参照日: 2026 年 1 月 7 日, 参照先: https://github.com/jhammond/xltop

RawlingsKen . (2016 年 5 月 11 日). Lustre Wiki - Lustre Monitoring Tool. 参照日: 2026 年 1 月 7 日, 参照先: https://wiki.lustre.org/Lustre_Monitoring_Tool

The Lawrence Livermore National Laboratory. (2025 年 7 月 9 日). GitHub - hpc/ior: IOR and mdtest. 参照日: 2026 年 1 月 7 日, 参照先: GitHub - hpc/ior: IOR and mdtest: https://github.com/hpc/ior/tree/main

IO500 Foundation. (2025 年 10 月 1 日). IO500. 参照日: 2026 年 1 月 7 日, 参照先: https://io500.org/

NVIDIA. (2025 年 12 月 5 日). 1. Benchmarking and Configuration Guide — GPUDirect Storage Benchmarking and Configuration Guide. 参照日: 2026 年 1 月 7 日, 参照先: https://docs.nvidia.com/gpudirect-storage/configuration-guide/

BreunerSven. (2025 年 12 月 29 日). GitHub - breuner/elbencho: A distributed storage benchmark for file systems, object stores & block devices with support for GPU. 参照日: 2026 年 1 月 7 日, 参照先: https://github.com/breuner/elbencho

UChicago Argonne, LLC. (2026 年 1 月 1 日). GitHub - argonne-lcf/dlio_benchmark: An I/O benchmark for deep Learning applications. 参照日: 2026 年 1 月 7 日, 参照先: https://github.com/argonne-lcf/dlio_benchmark

3.5施設・電力・冷却設計

本節では富岳NEXTにおける構築・設置・インテグレーションについて、基本設計段階における検討結果および設計方針を整理し、詳細設計に引き渡すための前提条件および制約を明確化する。特定の実装方式や製品選定に踏み込むのではなく、基本設計として建屋等の設置環境条件で確定すべき設計方針および検討上の前提を示す。

本報告書で扱う主な対象範囲は以下のとおりである。

- 計算機および周辺機の電力および電源条件の基本方針
- 計算機および周辺機の冷却条件の基本方針
- 計算機および周辺機のラック仕様の基本方針
- 計算機および周辺機を設置するためフリーアクセス要否、防塵、防火等の条件の基本方針
- 推定した各仕様を元に設置面積、床荷重などの諸条件の検討状況

以下の事項については本節の対象外とする。

- 計算ノードの仕様(CPU/GPU の仕様、発熱密度、電源容量、Scale-up/Scale-out ネットワークトポロジー)
- フロントエンド計算機（ログインノード、ウェブインターフェース用ノード、ファイル転送ノード、管理系サーバ群など）、第二階層ストレージ装置、周辺ネットワーク機器（外部接続ネットワーク、管理ネットワーク、制御ネットワーク、ストレージネットワーク）の仕様

以下の事項については本報告書の対象外とし、詳細設計以降の検討に委ねる。

- 計算機および周辺機のラック仕様、電源装置、電源分配器、冷却水分配器などの詳細仕様、具体的な機器構成
- 計算機および周辺機ラックと建屋との間の配電、冷却水配管の仕様

本節では、富岳NEXTの稼働開始2030年を見据えた高い演算性能と低環境負荷を実現できる計算機ラック仕様を検討するために、ラック電力密度、電源条件、冷却条件などに焦点を当て、「富岳」以降に稼働したHPCシステム製品およびOpen Compute Project (OCP) の動向調査を行い、実現のための課題抽出を行う。

3.5.1 現状分析と制約事項、課題

3.5.1.1 HPC システム/OCP 動向調査

3.5.1.1.1 HPC システム、OCP ラック動向調査

富岳NEXTでは高い演算性能を実現するため、高性能CPU、GPUを高密度に実装することが重要であり、これらを収納するラックあたりの電力は高くなることが想定される。そのラックの電力密度、電源条件、冷却条件など関する現状を把握するため、以下の「富岳」以降に稼働したHPCシステム製品およびOCPの動向調査を実施した。

- HPC システム製品：HPE Cray Supercomputing EX シリーズ、Lenovo Neptune、Atos Bull Sequana、NVIDIA MGX NVL72
- OCP ラック：OCP ORv3 HPR(High Power Rack)

調査の結果、以下の主要な傾向が確認された。

電力密度： ラックあたりの電力は既に100kWを超え、OCPラックでは今後300kW、さらには1MWの開発が計画されている。

電源条件： 電力供給能力の高いAC 3相480Vでの受電が主流である。また、ラック内のDC配電での電力損失を低減するため、DC48V以上の高い電圧が用いられている。さらに、OCP ORv3 HPR v4などおよそ400kW以上（一部例外があるが）のラックではACとDC50Vの間にDC±400VなどのHVDCを導入し、さらなる配電損失の低減を検討している。

冷却条件： 建屋からの冷水が32℃以上といった高水温での受入れを前提とし、かつ水冷比率が85%以上と高効率な水冷技術の採用が主流となっている。Direct Liquid Cooling (DLC) 用の冷却水供給装置である Coolant Distribution Unit (CDU) には、冷却対象の計算機ラック列に併設する In-Rowタイプ と、ラック内に設置する In-Rackタイプ がある。具体例として、約400kWと消費電力の大きい HPE Cray EX4000 には In-Rowタイプが、約140kWの HPE Cray EX2500 には In-Rackタイプがそれぞれ採用されており、システムの発熱量に応じて適切な方式が選択されている。

ラック床荷重：Atos BullSquana XH2000、Lenovo ThinkSystem SC750 V4 Neptuneでは2000kg/m2以上の高い床荷重となっている。

		HPE Cray		NSCC ASPIRE 2A [2][3]	Lenovo ThinkSystem SC750 V4 Neptune [4]	Atos BullSequana		NVIDIA MGX NVL72 [7]	OCP ORv3			
		EX4000 [1]	EX2500 [1]			XH3000 [5]	XH2000 [6]		HPR v1 [8]	HPR v2 [9]	HPR v3 [9]	HPR v4 [9]
Rack Size [mm]	H	2489	2302	–	2011	–	2020	(2300)	(2300)	–	–	–
	W	1181	900		600		750	600	600			
	D	1740	1719		1200		1270	1200	(1223)			
Weight[kg]		3629	1462	–	1836	–	2035	–	–	–	–	–
Floor load [kg/m^2]		1766	945	–	2550	–	2136	–	–	–	–	–
# of Blade		64	24	–	–	~ 38	32	18	–	–	–	–
Power/Rack(max)		400kVA @480V	141kVA @480V	–	–	120kW	90kW	120kW	–	190kW	300kW	800kW ~1MW
Power Supply (Up to power delivery to the nodes)		AC480V DC380V	AC480V DC380V	–	AC480V DC48V	–	AC400V DC**	AC480V DC50V	AC480V DC50V	AC480V DC50V	AC480V DC50V	AC480V DC±400V DC50V
Cooling Conditions Inlet Water temp.[℃] Inlet Air temp.[℃] Liquid-cooling ratio Cooling water ΔT		W:32 A:– 100% –	W:32 A:– 100% –	W:40 A:– 100% –	W:45 A:40 100% –	W:40 A:– 97% –	W:40 A:– 95% –	W:45 A:35 85% –	–	–	–	–
CDU type		In-Row	In-Rack	In-Row	In-Rack / In-Row	In-Rack	In-Rack	In-Rack / In-Row	–	–	–	–

[1] HPE, “HPE Cray Supercomputing EX QuickSpecs”, https://www.hpe.com/us/en/collaterals/collateral.a00094635enw.html

[2] National Supercomputing Centre Singapore (NSCC), “ASPIRE2A General QuickStart Guide”, https://help.nscc.sg/wp-content/uploads/2024/06/ASPIRE2A-General-Quickstart-Guide-1.pdf

[3] National Supercomputing Centre Singapore (NSCC), “NUS NSCC i4.0 DC A Tropical Supercomputing DC”, https://www.nscc.sg/wp-content/uploads/2022/07/NUS-NSCC-i4.0-DC-V1.0-2.pdf

[4] Lenovo, “Lenovo ThinkSystem SC750 V4 Neptune Server Product Guide”, https://lenovopress.lenovo.com/lp2009-thinksystem-sc750-v4-neptune-server.html

[5] Atos, “Evolution of the Sequana System Architecture”, https://indico3-jsc.fz-juelich.de/event/149/contributions/603/attachments/178/256/2024_05_24_Evolution-of-the-Sequana-System-Architecture____Friedrich.pdf

[6] Atos, “BullSequana XH2000 Data Sheet”, https://atos.net/wp-content/uploads/2020/07/BullSequanaXH2000_Features_Atos_supercomputers.pdf

[7] NVIDIA, “MGX Accelerated Computing Rack and Trays Specification Revision 1.0”, https://www.opencompute.org/documents/mgx-accelerated-computing-rack-and-trays-specification-101024-pdf

[8] OCP, “ORv3 High Power Specifications”, https://drive.google.com/drive/folders/1t39vYwhEwFmjXH_bXEdlJ2AFnubRnj7H

[9] Meta, “ORv3 HPR “Next” Rack and Power Solution for AI Systems”,

https://www.dropbox.com/scl/fi/q7d9pjvghet5urai2uo68/6529-Metas-ORv3-HPR-Next-Ecosystem-Solution.pdf?rlkey=n7x63uhqsm9sq4znnu3nom04d&st=xejtidce&dl=0

3.5.1.1.2　Power Shelf/Power Supply Unit の水冷化について

OCP ORv3 High Power Rack（HPR）で規定されているPower Shelfおよびその内部に搭載されるPower Supply Unit（PSU）は、現行仕様では空冷方式を前提としている。今後のシステムにおいては、高密度化および高電力化の進展に伴い、電源部の発熱がシステム全体の制約要因となる可能性が高い。このため、電源を含めた水冷化は重要な検討課題であり、本検討では、PSUベンダーに対して水冷化に関する検討状況のヒアリングを実施した。

ヒアリングの結果、サーバーベンダーからPSUの水冷化に関する検討要請は存在するものの、PSU内部は高密度実装となっており、冷却対象となる部品が分散配置されていることが確認された。このため、クーリングプレートやヒートパイプを適切に配置するためのスペースは極めて限定的であり、冷却構造は複雑化する傾向にある。さらに、水冷化を実現するためには、電源容量の削減、あるいは筐体体積の増加が必要となる可能性が示された。これらの制約により、現時点では水冷PSUの製品化は限定的な状況にとどまっていることが明らかとなった。

詳細設計においては、技術的制約および実装課題を踏まえつつ、ベンダーと連携しながら水冷化に向けた検討を継続的に実施していく。

3.5.1.2　制約事項

本節では、富岳NEXTの設計を進める上で遵守すべき前提条件となる制約事項を明確にする。これらは、後続章における要求事項および設計検討において遵守すべき前提条件である。

3.5.1.2.1　計算ノード、周辺機全体の最大電力

富岳NEXTの運用における電力コストを抑制するため、以下の最大電力制約を設ける。

- 計算ノードおよびネットワークを含むシステム全体の最大消費電力は、運用時において 40MW 以下とする。一方で、機器構成に基づく設計上のピーク電力については 40MW を超えることを許容する。運用時において計算ノードやラックなどで負荷を制約する電力管理機構、電力制御機能等を用いることで、システム全体の消費電力を 40MW 以下に抑制する運用を想定している。
- 周辺機器(ログインノード、ストレージ、管理系サーバ及び外部接続スイッチなど)の最大電力は 2MW 以下とする。

3.5.1.2.2　計算ノードの冷却条件

環境負荷の低減と運用コストの最適化を目的として、以下の冷却条件を富岳NEXTの設計に課す。

- 計算ノードの構成要素は、そのほとんど全てが水冷方式を採用することを基本とする。
- チラーを介さずに、外気等を利用したフリークーリングによる冷却条件を満たすこととする。
- 建屋側から供給する冷却水のみを使って冷却するとともに、圧縮機を使った冷却機器は用いないこととする。

3.5.1.2.3　計算ノード、周辺機全体の設置面積

インフラ投資の抑制と効率的なスペース利用を図るため、以下の設置面積制約を設ける。

- 計算ノードを設置するスペースは、500 ㎡程度に区切られた部屋が 4 つ準備されるものとする。
- 周辺機器を設置するスペースは、500 ㎡程度に区切られた部屋が 1 つ準備されるものとする。

この制約の下、システム設計においては、可能な限り少ない設置面積で高密度にコンポーネントを配置し、スペー

ス効率の最大化を追求することが求められる。

3.5.1.3 課題

HPCシステムおよびOCP動向調査、並びに明確にした富岳NEXTの設計における制約事項を踏まえ、高い演算能力の実現、環境負荷低減、および運用コストの最適化を達成するための課題を示す。

3.5.1.3.1 高密度電力供給と効率的な配電、電源技術の確立

現状のHPCシステムはラックあたり100kWを超える電力密度に達しており、将来的に300kW、さらには1MW級の電力供給が求められる。これに対し、以下の課題が存在する。

- 高い電圧の採用による給配電の効率化：ラック全体電力の増大に伴う AC 入力系統の増加を抑えるため、AC 3 相 400V 以上の受電技術の採用が必須である。また、電力増加によって上昇する電流量の増加に伴い、計算ノードへの配電経路の導体抵抗などによる発熱、さらに配電経路の導体温度上昇による導体抵抗の増加率なども無視できなくなる。各コンポーネントの電圧を高くするとともに、損失、または、その発熱を低減させ、電流密度を高めることが重要である。
- Power Shelf/PSU の変換損失による排熱の低減: ラック全体電力の増大に伴い、Power Shelf 内の PSU 内部の電圧変換による損失、およびその損失による発熱も無視できなくなっている。このことから、高効率（低損失）な PSU の採用が極めて重要である。あわせて、現状では水冷 PSU は一般的にはなっていないが、水冷化技術の採用についても継続調査する。

3.5.1.3.2 高発熱への対応を可能とする冷却システムの実現

ラックの高密度化に伴う大きな発熱量を効率的に冷却することが求められる。これに対し、以下の課題が存在する。

- コンポーネントレベルでの水冷化の徹底: CPU、GPU、メモリ、ストレージデバイス、ネットワークインターフェースなど、可能な限りの発熱部品に対し水冷技術を適用し、システム全体の水冷比率を最大化させることを基本方針とする。その上で、水冷化に伴う製造コストは設計上の制約条件とはせず、運用コストや信頼性、将来の拡張性を含めたライフサイクル全体の観点から評価し、最適な構成を検討する。
- 高温水冷採用: 大きな電力を必要とする計算ノードなどに低い温度の冷却水を用いることはチラーの稼働を高め、エネルギー効率や環境負荷の観点で不利となる。そのため、環境負荷の低減、運用コスト最適化の観点から、チラーを必要としないフリークーリングを前提とした温水冷却の採用が重要である。さらに、部品温度の上昇が信頼性に与える影響を考慮し、十分な温度マージンの確保や評価を行ったうえで、高水温条件下においても安定した運用が可能な設計とする必要がある。
- 低消費電力時の PUE 悪化抑止：HPC システムでは高負荷時とアイドル時で電力消費の変動が大きくなる。この変動に対応できない冷却システムは、低消費電力時に過剰な冷却を行い、PUE を悪化させる原因となる。そのため、電力変動に応じた冷却能力の最適化を行うなど、建屋や CDU と計算機のコデザインが必要となる。CDU の流量制御機能を採用することに加え、CDU から計算ノードへの冷却水供給部での負荷状況に応じた流量制御を可能にするため、バルブ調整機構を装備することも重要である。

3.5.1.3.3 高い設置効率の実現

コスト最適化の観点から、設置面積の適正化、最小化が求められる。これに対し、以下の課題が存在する。

- 高密度と保守性を両立したラックの選定：高密度実装はラック間接続ケーブルの本数も増大させ、保守作業性を悪化させることにつながる。よって、高密度実装を維持したまま保守性を向上させる最適化や、通信、電源、水冷配管などの効率的なレイアウト検討をすることが重要である。
- 高床荷重への対応：ラックの高密度化により、床荷重も増加する。そのため、部屋の床耐荷重の強化および荷重の分散、適切な耐震対策が必要である。また、高重量ラックは、ラック搬入時の安全性も重要となる。具体的には、搬入経路における、開口部寸法、耐荷重、段差の有無などの検討が必要である。この際、ラック搭載機器を搬入後に搭載することで、搬入時の総重量を低減する方策も検討に含めるべきである。

3.5.2 要求事項

富岳NEXTの基本設計を進める上で、システム全体および建屋含めた各構成要素が満たすべき要求事項を明確にする。以下の要求事項は、計算ノード設置部、周辺機器設置部、およびそれらの共通事項の3つのカテゴリに分類し、それぞれ具体的な数値目標や条件を提示する。これらの要求事項は、今後のシステム設計、機器選定、インフラ構築の根拠となる。

3.5.2.1 計算機設置部への要求事項

計算機設置部には計算ノード、Scale-upネットワークスイッチ、Scale-outネットワークスイッチ、管理・制御ネットワークスイッチ、これらに冷媒を供給するCoolant Distribution Unit(CDU)、電源装置を設置する。富岳NEXTの演算性能目標に加え、環境負荷の低減と運用コストの最適化に向けた要求事項を以降に示す。

3.5.2.1.1 電力、電源条件

- 計算ノード負荷を制約する電力管理機構などの機構を活用し、計算ノードおよびネットワーク全体の最大消費電力が 40MW を超過しない運用を実現することを要求する。
- 計算機設置部に設置する装置、機器は AC3 相 4 線式 415V 又は 480V の受電に対応することを要求する。
- ラック内の DC 配電での電力損失低減のため、ラック内配電は定格 DC48V 以上とすることを要求する。また、配電経路での損失や、それに伴う発熱による電流密度の悪化を抑制できるよう、適切な導体、導体形状、冷却構造の採用を要求する。

3.5.2.1.2 冷却条件

- 環境負荷低減、運用コストの最適化のため、冷却方式はチラーを使用しないフリークーリングを前提とする。この冷却方式において、冷却水条件は ASHRAE W4 (W45)、空調条件は ASHRAE A4 とする。設置する装置および機器は、これらの条件下でも動作可能であることを要求する。ただし、富岳 NEXT を設置する神戸では真夏以外は ASHRAE W32 を満足できることを前提として、ASHRAE W32 ではシステム全体の最大消費電力 40MW が動作可能、ASHRAE W32 を超える際にはシステムの動作台数の低減することも許容する。
- 計算ノードの消費電力の 95%以上を Direct Liquid Cooling とすることを目標に検討することを要求する。
- 冷却後の排熱利用を前提とし、建屋からの入水温と出水温の差を 10 度程度になるよう、電力に応じた流量制御機構を設けることを要求する。

3.5.2.1.3 設置面積、床荷重、ラック構造

- 計算機設置部として 500 ㎡程度の区切られた部屋を 4 つ提供されることを前提とし、このスペースにおいて、省スペース化を追求し、さらに運用コストや効率性を考慮した上で使用部屋数を最適化することを要求する。
- ラックの床荷重は 3000kg/㎡以下になるように検討を行い、超える場合は荷重分散を行うことを要求する。フリーアクセスは不要することを基本方針とする。
- ラックは物理的、冷却的、電源供給観点で高密度実装可能であり、かつ、ケーブルの集積度が高まる状況下でも、確実にケーブルの抜き差しや配線の確認が出来るよう、十分な保守作業性を確保できる適正

なラックを選定することを要求する。また、高コストにならないよう OCP で制定されている標準規格などの出来る限りオープンな規格を採用することを要求する。

- 計算機設置室内での保守作業時間を最小化することを目的として、交換作業が容易な構造とすることを要求する。加えて、作業時間を要する部品およびユニットについては、別室（居室）において作業可能となるよう、可搬性を考慮した構造とすることを要求する。
- 部屋への入室者が意図的、不注意に関わらず、不用意に接触することを防止できるよう、各ラックには鍵付きの扉を設けることを要求する。

3.5.2.2 周辺機設置部への要求事項

周辺機設置部にはフロントエンド計算機（ログインノード、ファイル転送ノード、管理系サーバ群など）、第二階層ストレージ装置、周辺ネットワーク機器（外部接続ネットワーク、管理ネットワーク、制御ネットワーク、ストレージネットワーク）を設置する。周辺機においても目標の処理能力を達成するとともに、環境負荷の低減と運用コストの最適化に向けた要求事項を以降に示す。

3.5.2.2.1 電力、電源条件

- 周辺装置(ログインノード、ストレージ、管理系サーバ及び外部接続スイッチなど)の最大運用電力は 2MW 以下とすることを要求する。
- バックアップ電源で保護されたものが供給されることを前提に、周辺機設置部に設置する装置、機器は 3 相 4 線式 415V の受電に対応することを要求する。

3.5.2.2.2 冷却条件

- 周辺機器自体は空冷で、1 ラックあたり 30kW 程度を前提とし、ASHRAE W1 の条件で排熱を Rear Door Heat Exchanger(RDHx、リアドア)で取り除くことを要求する。
- 空冷条件は ASHRAE A1 とし、この条件で冷却できる装置、機器の選定を要求する。

3.5.2.2.3 設置面積、床荷重、ラック構造など

- 周辺装置設置部として 500 ㎡程度の区切られた部屋を 1 つ提供されることを前提とし、このスペース内に可能な限り、省スペースで収納することを要求する。
- ラックの床荷重は 3000kg/㎡以下になるように検討を行い、超える場合は荷重分散を行うことを要求する。フリーアクセスが必要な場合は計算機設置側の負担で必要な範囲の床及び搬入用スロープなどを設置できる。
- ラックは装置、機器を高密度実装可能、また、ケーブル密度増でも保守作業性を確保するため、EIA/ECA-310-E 19 インチラックの 48U 程度高さのラックとし、適正な幅、奥行きを持つラックを選定することを要求する。
- 部屋への入室者が意図的、不注意に関わらず、不用意に接触することを防止できるよう、各ラックには鍵付きの扉を設けることを要求する。

3.5.2.3 計算機/周辺機設置部共通の要求事項

3.5.2.3.1 設置、保守作業員の労働安全衛生確保

保守作業員の労働安全性については、保守作業員の装備等を充実するとともに、計算機設置部と実際に保

守する居室を分割することや作業時間を制限することで確保することを要求する。

3.5.2.3.2　防塵、防火

防塵・防火については、各種法令や下記に記載する条件が満たされることが望ましいが、一方でシステムの導入コストや運用コストを踏まえ、諸条件を検討する。

- 計算機設置部、周辺機設置部に設置する装置、機器の浮遊塵埃による物理的損傷、冷却効率の低下、ショート・腐食の誘発を避けるため、浮遊塵埃基準「ISO 14644-1 クラス 8」の清浄度を満たすことが望ましい。
- 計算機設置部、周辺機設置部に設置する装置、機器は万が一の発火の延焼、および周囲の火災発生時における装置、機器の着火からの延焼を防ぐため、「IEC62368-1：オーディオ/ビデオ、情報および通信技術機器－安全要求事項」に準拠した防火エンクロージャを有することが望ましい。

3.5.3 システム諸元

本節では、要求事項について、計算機設置部、周辺機設置部それぞれについての設置形状、床荷重、設置面積、冷却/電源条件の検討結果を示す。

計算機設置部に設置する本体システムについて、富岳NEXTの演算性能目標を達成させるために、アーキテクチャ検討会およびコデザイン検討会において、2つのCPU、4つのGPUからなる計算ノードを、Scale-upネットワークスイッチを用いて接続し、Scale-outネットワーク構成として4つのレールを有するRail-optimized fat-tree（Quad-rail optimized fat-tree）トポロジで約3,400ノードを接続するリファレンス構成が検討された。このリファレンス構成はアーキテクチャ検討を目的として用いられたものであり、最終的なシステム仕様を示すものではない。

2026年2月時点では計算ノード内に搭載する第一階層ストレージのフォームファクターには、EDSFF E1.SとEDSFF E3.Sの２案があり、EDSFF E1.Sを採用する場合には計算ノードが1RUとなるが、許容消費電力が大きく出来るEDSFF E3.Sの場合には、計算ノードは2RUとなると推定しており、本書では2案を検討した。

3.5.3.1 設置形状

本項では計算機設置部、周辺機設置部に設置する主要なラックについて、サービスエリアを含めた設置形状を示す。本検討では前面に扉が取り付けられていないが、詳細設計にて前扉を含めた検討を行う。

3.5.3.2 計算ノードラック

計算機設置部に設置する計算ノードラックとしてNVIDIA社がOCP ORv3ラックをベースにOCP規格化したMGXラック(リアエクステンダ有)を想定する。計算ノード、Scale-upネットワークスイッチ、4RU Tier-1 Scale-outネットワークスイッチ、管理・制御ネットワークスイッチおよび電源(Power Shelf)を搭載する。電源(Power Shelf)は、OCP ORv3またはMGX rack向けに規定された3相4線415V又は480Vを受電可能なものを想定する。Power ShelfからDC48V以上の電圧で供給し、大電流容量のバスバーを介して計算ノードなどに配電することを想定する。

計算ノード、Scale-upネットワークスイッチ、Scale-outネットワークスイッチはDirect Liquid Cooling(DLC)とするが、管理・制御ネットワークスイッチ、電源(Power Shelf)は空冷と仮定して、これらの排熱を設置部屋に極力放出しないように、リアドア(Rear Door Heat Exchanger)にて排熱を回収することを本書では想定した。詳細設計にてIn-Rowタイプの熱交換機も含む複数の方式を対象に、各種要件を踏まえた最適な冷却方式の採用を継続検討する。仮検討したリアドアを含めたラックサイズはW600mm x H2300mm x D1482mmとなる。詳細設計の段階で建屋条件に最適なサービスエリア寸法の検討を行う。

3.5.3.2.1 Scale-out ネットワークスイッチラック

計算機設置部に設置するScale-outネットワークスイッチラックとしてOCP ORv3 HPRラックを想定。4RU Tier-2 Scale-outネットワークスイッチ、管理・制御ネットワークスイッチおよび電源(Power Shelf)を搭載する。電源(Power Shelf)は、OCP ORv3で規定された3相4線415V又は480Vを受電可能なものを想定する。Power ShelfからはDC48V以上の電圧で供給し、大電流容量のバスバーを介して4RU Tier-2 Scale-outネットワークスイッチに配電することを想定する。

Scale-outネットワークスイッチはDirect Liquid Cooling(DLC)とするが、管理・制御ネットワークスイッチ、電源(Power Shelf)は空冷と仮定して、これらの排熱を設置部屋に極力放出しないように、リアドア(Rear Door Heat Exchanger)にて排熱を回収することを本書では想定した。詳細設計にてIn-Rowタイプの熱交換機も含む複数の方式を対象に、各種要件を踏まえた最適な冷却方式の採用を継続検討する。仮検討したリアドアを含め

たラックサイズはW600mm x H2300mm x D1482mmとなる。

3.5.3.2.2　周辺機ラック

周辺機はEIA/ECA-310-E 19インチラックに搭載可能なものから選定する。周辺機の高密度化および0U PDU採用によるケーブル密度の増加が見込まれるが、アーキテクチャ検討会および周辺機で検討したケーブル本数が仮に大幅に増加しても収納できる幅800mm、奥行き1200mm、高さ48Uのラックを想定した。周辺機に配電する0U PDUは3相4線式415Vを受電可能なものを詳細設計にて選定する。周辺機の排熱を回収するリアドアを含めたラックサイズはW800mm x H2300mm x D1482mmとなる。

3.5.3.3　床荷重

計算機設置部および周辺機設置部それぞれにおいて、ラック密度が高いと推定される計算ノードラックとHDDを使用した第二階層ストレージラックの床荷重の検討結果を示す。それぞれ最大2,302kg/m2、1,381kg/m2となり、3,000kg/m2を超えない結果となった。詳細設計にて継続検討を行い、仮に3000kg/m2を超える場合は、搭載密度の低減、もしくは鉄板敷設などによる荷重分散を検討する。

3.5.3.3.1　計算ノードラック

計算ノードラックとしてMGXラック(NVL72)を仮定し、その質量の推定のためAivres KRS8500V3 (GB300 NVL72) および nVent Rack Chiller RDHX PRO Activeの数値を用い、2RU計算ノードの質量は1RU品の約1.2倍、Power Shelfは33kWから72kW化への高出力化に伴い約1.4倍を仮定し概算を行った。

3.5.3.3.2　第二階層ストレージラック

周辺機器の選定は2027年から2028年に実施される予定のため、本書では周辺機の中で質量の大きいと想定される第二階層ストレージのHDDストレージについて、HPE Cray ClusterStore E1000 SSU-D(4U106)、nVent Rack Chiller RDHX PRO Activeの値から概算を行った。

3.5.3.4　設置面積

3.5.3.4.1　計算機設置部

計算機設置部 500m2程度の区切られた4つの部屋に設置できるかの検討を下記想定で行い、ラック数の多い案2でも1部屋当たり約203m2となり、設置可能な見込みである。詳細設計では、建屋の柱や分電盤などの周辺機の配置を考慮した上で、設置する部屋数の削減の検討も行う。

[想定条件]

- 500m2 の部屋の寸法は横 25m、縦 20m
- 計算機設置部に搭載するラックを 4 つの部屋に均等割り
- 1 島の配置は、In-Row CDU 2.1MW(*)で冷却できるラック数で仮置き (*2025 年時点で発表されているもので冷却能力の大きい Lite-on 社製の CDU を仮設定)
- 建屋の柱、建屋入退室扉開閉部などの設置出来ないエリアや周辺機（分電盤など）は含まず
- 各ラックのサービスエリアを反映し、ラック前面側は 1.8m、背面側は 1.2m、各島の左右には 1.4m のサービスエリアを設けて、下記のラック構成を想定して検討中である。

	案 1	案 2
計算ノードラック	36 台	54 台
Scale-out ネットワークスイッチラック	3 台	3 台
In-Row CDU ラック	8 台	8 台
設置面積	12m x 13.5m = 162m^2	15m x 13.5m = 202.5m^2

3.5.3.4.2　周辺機設置部

周辺機設置部 500m2程度の区切られた1つの部屋に設置できるかの検討を下記想定で行い、約441m2となり、設置置可能な見込みである。詳細設計でラック数の削減検討を行い、また、建屋の柱や分電盤などの周辺機の配置を考慮した検討を行う。

[想定条件]

- 500m2 の部屋の寸法は横 25m、縦 20m
- 1 島は最大 10 ラック
- 建屋の柱、建屋入退室扉開閉部などの設置出来ないエリアや周辺機（分電盤など）は含まず
- 周辺機ラックのサービスエリアを反映し、ラック前面側は 1.8m、背面側は 1.2m、各島の左右には 1.4m のサービスエリアを設けて、下記ラックを配置

● 周辺機ラックの台数は下記で仮置き

フロントエンド計算機ラック	2 台
第二階層ストレージラック（データ部）	84 台
第二階層ストレージラック（メタデータ部）	12 台
ストレージネットワークラック	3 台
管理・制御ネットワークラック	2 台
設置面積	22.4m x 19.7m = 441.3m^2

第二階層ストレージについては必要な設置面積が大きいFlash+HDD構成を採用した。

3.5.3.5　消費電力

3.5.3.5.1　計算機設置部

アーキテクチャ/コデザイン検討会にてCPU/GPU/Scale-up/Scale-outネットワーク機器などの主要部品の電力の推定、ヒアリングを行っている。最大電力は40MWを超えないよう、計算ノード及びネットワークのシステム規模を2027年に決定し、かつ、稼働台数の制限やパワーキャッピング機能の活用など運用技術により対応する。仮に全体システムを４つの部屋に均等に設置し、稼働台数制限による電力制限を行った場合でも40MW上限まで稼働できるよう、各部屋の上限は40MWの均等割り(＝10MW)に制限しないよう、詳細設計期間内で継続検討を行う。

3.5.3.5.2　周辺機設置部

周辺機設置部に設置するフロントエンド計算機、第二階層ストレージ、周辺ネットワーク装置、それら排熱を冷

却するリアドアを含めたトータルの最大電力は2MW以下、ラックあたりの最大電力は30kW程度となるよう、機器選定含めて、詳細設計において運用システム・利用環境整備、ストレージなどの検討会にて検討を行う。

3.5.3.6 冷却条件

3.5.3.6.1 計算機設置部

計算機設置部に設置する装置、機器は、冷却水条件ASHRAE W4(W45)および空調条件ASHRAE A4で動作可能となるよう詳細設計を行う。設置する神戸の気象条件を考慮し、真夏を除く期間はASHRAE W32の条件下で最大電力40MWまで稼働させ、それを超える場合は稼働台数制限を行うことなどを詳細設計にて検討をする。

計算ノードラックの水冷比率について検討を行った。計算ノード、Scale-up/Scale-outネットワークスイッチ、ラック内のDC48V(DC50V)を配電するバスバーは完全水冷を前提とする。それ以外の管理・制御ネットワークスイッチ、Power Shelfは空冷、Power Shelfの損失(効率)を3.5%(96.5%)と仮定した。電力の大半を占める計算ノードを完全水冷、高効率電源を想定することにより、水冷比率は約97%となった。この場合でも空冷電力は10kWを超えている。詳細設計にて、リアドアやIn-Rowタイプの熱交換機も含む複数の方式を対象に最適な冷却方式を検討するとともに、Power Shelf、管理・制御ネットワークスイッチの水冷化の検討をアーキテクチャ検討会と連携して行う。

システム全体の最大消費電力40MW時には、建屋から供給される冷却水と建屋に戻す冷却水との間に10℃の温度差を維持するため、約58,010 LPM（リットル/分）の流量が必要となる。一方、システムの待機時（アイドル時）の消費電力が30%の12MWと仮定した場合、流量制御しなければ、冷却水の温度差が3℃程度となる。廃熱活用などの観点から、待機時が長く続く場合においても建屋冷却水の入口・出口の温度差を10℃程度に維持可能な流量制御方式について、詳細設計で検討する。この検討においては、CDUに備わる流量調整機構を活用する。さらに、計算ノードにバルブ等を設置し、個別に流量調整機能を持たせる可能性についても検討する。

以下に計算機設置部に設置する機器の設置条件を示す。

計算機設置部水冷条件

項目		仕様	備考
1 次側冷却水温度		ASHRAE W4(W45)	
1 次側流量		3045LPM 以上/CDU1 台	CDU の冷却能力最大 2.1MW 時
1 次側最大圧力損失		0.2 MPa (T.B.D)	
1 次側最大許容圧力		1 MPa (T.B.D)	
1 次側冷却水 ΔT		約 10℃	
1 次側配管（カプラ）径		3 インチ(T.B.D)	
1 次側冷却水条件			
	BTA 濃度	10~100ppm	
	pH	7~9	
	導電率	100μS/cm 以下	

計算機設置部設置機器の空冷条件

項目	仕様	備考
供給空気温度	ASHRAE A4	
湿度	ASHRAE A4	
露点	(T.B.D)℃以下	結露しないこと
風量	(T.B.D) m^3/min/ラック	
排出空気温度	(T.B.D)～(T.B.D)℃	
空気 ΔT	(T.B.D)℃	

3.5.3.6.2　周辺機設置部

周辺機器設置部の空冷条件ASHRAE A1はICT機器としては一般的な条件であるため冷却可能な見込みである。その全体の排熱を30kW程度とした場合、nVent社のリアドア(RDHX PRO Active)では冷却水が24℃、排気温度が27℃の場合、44kWの冷却能力があるため、冷却水条件ASHRAE W1で冷却可能な見込み。詳細設計期間にて適切な機器の選定を行う。以下に水冷、水質、空調条件を示す。

周辺機設置部水冷条件

項目		仕様	備考
1 次側冷却水温度		ASHRAE W1	
1 次側流量		43.1LPM 以上/ラック 2873LPM 以上/部屋	30kW/ラック時 2MW/部屋時
1 次側最大圧力損失		0.5 MPa (T.B.D)	
1 次側最大許容圧力		1MPa (T.B.D)	
1 次側冷却水 ΔT		約 10℃	
1 次側配管（カプラ）径		1.5 インチ(T.B.D)	
1 次側冷却水条件			
	BTA 濃度	10~100ppm	
	pH	7~9	
	導電率	100μS/cm 以下	

周辺機設置部設置機器の空冷条件

項目	仕様	備考
供給空気温度	ASHRAE A1	
湿度	ASHRAE A1	
露点	(T.B.D)℃以下	結露しないこと
風量	(T.B.D) m^3/min/ラック	
排出空気温度	(T.B.D)～(T.B.D)℃	
空気 ΔT	(T.B.D)℃	

3.5.3.7 防塵、防火

- 浮遊塵埃基準：ISO 14644-1 クラス 8 の清浄度は、ICT 機器としては一般的な基準であるため、遵守可能の見込み。本基準を元に詳細設計を進める。
- 防火対策としての「IEC62368-1：オーディオ/ビデオ、情報および通信技術機器 – 安全要求事項」への準拠は可能な見込み。本基準に元に詳細設計を進める。

3.5.3.8 電源条件

本項では計算機設置部、周辺機設置部に設置する主要なラックの電源条件および電源系統図を示す。

- 相数　　：3 相
- 電圧　　：AC415V±10%(3 相 4 線)又は AC480V±10%(3 相 4 線) [計算機設置部]
  ：AC415V±10%(3 相 4 線) [周辺機設置部]
- 周波数　：50/60Hz
- 瞬断耐力：T.B.D (詳細設計で検討を行う)

電圧は周辺機設置部では、一般的な周辺装置の採用を前提とし、AC単相240Vが供給可能なAC3相4線415Vとする。一方、計算機設置部については AC3相4線415V、480Vのどちらでも対応可能であり、2026年度中に決定を行う。

3.5.4 建屋等に対する設置環境条件

本節では建屋に関する設置環境条件を示す。

3.5.4.1 床荷重

設置するラック当たりの高密度化、設置スペース削減による環境負荷低減、運用コスト最適化を実現容易にするために、床耐荷重は3000kg/m2以上と設定する。

3.5.4.2 防塵

設置する装置、機器の浮遊塵埃による物理的損傷、冷却効率の低下、ショート・腐食の誘発を避けるため、計算機設置部、周辺機設置部は浮遊塵埃基準：ISO 14644-1 クラス 8の清浄度を満たすこと。

3.5.4.3 電源設備

設置する装置、機器の安定稼働のため、電源設備に求められる設備条件は以下の通り。

- 相数　　：3 相
- 電圧　　：AC415V±10%(3 相 4 線)又は AC480V±10%(3 相 4 線) [計算機設置部]
  AC415V±10%(3 相 4 線) [周辺機設置部]
- 許容電圧変動　：入力電圧に対し±10%以内
- 電源周波数　　：60Hz
- 電源周波数変動　：定格周波数±3Hz 以内

3.5.4.4 冷却設備

設置する装置、機器の安定稼働のため、冷却設備に求められる設備条件は以下の通り。

計算機設置部、周辺機設置部の冷却水条件

項目		計算機設置部	周辺機設置部
1 次側冷却水温度		ASHRAE W4	ASHRAE W1
1 次側流量		58010 LPM 以上/システム @40MW	2873 LPM 以上/部屋 @2MW
1 次側冷却水条件			
	BTA 濃度	10~100ppm	
	pH	7~9	
	導電率	100μS/cm 以下	

周辺機設置部の空調条件

項目	仕様	備考
供給空気温度	ASHRAE A1	
湿度	ASHRAE A1	

3.5.5 検討事項のまとめ

基本設計で確定した主要な項目、今後の検討が必要となる詳細設計における主要な検討事項、および他WG・建屋設計への連携事項について記述する。

3.5.5.1 基本設計での確定事項

本節では、基本設計段階において既に確定した事項を示す。これらは詳細設計の前提条件である。

- 計算機および周辺機の電力上限は、計算機設置部では最大 40MW、周辺機設置部では 2MW とし、その実現のために電力管理機構などを用いることとする。
- 計算機設置部は AC3 相 4 線 415V 又は 480V、周辺機設置部は AC3 相 4 線 415V を受電可能とし、電力密度の高い計算ノードラック内の DC 配電を 48V 以上の高圧とすることとする。
- 計算機はチラー不要なフリークーリング方式での冷却を前提とし、冷却水条件 ASHRAE W4(W45)、空調条件 ASHRAE A4 で動作可能とする。ただし、設置地である神戸での気象条件を勘案し、ASHRAE W3 で最大消費電力 40MW を冷却可能とし、W3 を超える場合には稼働制限を行うことも許容する。
- 計算ノードの大部分を Direct Liquid Cooling とすることとする。
- 周辺機は空調条件 ASHRAE A1 で空冷可能であり、その排熱をリアドアで冷却水条件 ASHRAE W1 で回収可能なものを選定することとする。
- 計算機および周辺機のラックは設置面積削減のため、高密度実装可能なものとすることとする。
- 計算機および周辺機を設置するためフリーアクセスは不要を前提とすることとする。
- 計算機および周辺機の安定稼働、防火のため、一般的な浮遊塵埃基準、安全規格に準拠した防火エンクロージャを有することとする。
- 推定した各仕様からは、設置面積および床荷重は建屋の仕様内であること確認済みとする。

3.5.5.2 詳細設計での検討事項

本節では、詳細設計段階で検討を進めるべき事項を示す。

- 計算機および周辺機のラック仕様、電源装置、電源分配器、冷却水分配器などの詳細仕様、具体的な機器構成
- 計算機および周辺機ラックと建屋との間の配電仕様および冷却水配管の仕様
- 計算ノードラックの空冷電力の削減、空冷排熱の最適な冷却方式
- CDU の冗長方針および具体的な構成
- 高負荷時、待機時（アイドル時）の電力変動に応じた流量制限機構。CDU による流量調整機構を基本とし、計算ノードへの個別流量制御機能（バルブ等）追加の可能性検討
- 周辺機ラックのケーブル本数、保守性を考慮したラック幅、奥行き寸法の最適化
- 建屋条件に最適なサービスエリア寸法の検討
- 計算機設置部および周辺機設置部での作業者の労働安全衛生の確立。特に、室内が高温になる環境での保守交換作業について、作業の時間短縮に繋がる施策や、長時間作業が必要な際に設置場所とは別の部屋で作業が行えるような方法を検討
- 製品製造時の CO2 排出量算出、その精度向上策

3.5.5.3　他 WG・建屋設計への連携事項

本節では、詳細設計段階で検討を進めるべき事項を示す。

3.5.5.3.1　アーキテクチャ検討会への連携事項

- 温水冷却で、かつ、計算ノードの消費電力の 95%以上を Direct Liquid Cooling(DLC) とすることを目標とする
- 計算ノードの入力定格は DC48V 以上とする
- Power Shelf、管理・制御ネットワークスイッチの DLC 化の検討、および Power Shelf、ラック内配電のバスバーの損失低減や、それに伴う発熱低減に関する検討についてアーキテクチャ検討会と連携して推進する。
- 本検討会と連携して高負荷時、待機時（アイドル時）の電力変動に応じた流量制限機構の検討を行う。併せて、計算ノードにバルブ等を設置し、個別に流量調整機能を持たせる可能性についても検討する。
- 計算機設置部での作業者の労働安全衛生の確立のため、特に、室内が高温になる環境での保守交換作業について、作業の時間短縮に繋がる構造や、長時間作業が必要な際に設置場所とは別の部屋で作業が行えるような構造について検討する。

3.5.5.3.2　ストレージ検討会、運用システム・利用環境整備検討会への連携事項

- 空冷条件 ASHRAE A1 で冷却可能、AC 単相 240V で受電可能な装置の選定を行う。
- 48U の 19 インチラックの採用を前提に本検討会と連携して設置スペースの適正化に向けたラック搭載構成の検討を行う。

3.5.5.3.3　建屋設計への連携事項

- 床荷重、防塵、電源設備、冷却設備の各条件を建屋設計へ確実に反映する。

3.5.6 データセンターのデジタルツイン

富岳NEXTにおける建屋および計算機の設計・設置・運用を一体的に最適化するため、データセンターのデジタルツインを構築し、設計および運用に活用することを検討した。高密度化・高電力化が進む次世代HPC/AIシステムにおいては、電力、冷却、配置、運用負荷の相互依存関係が複雑化しており、従来の静的な設計検討のみでは最適化が困難である。このため、実環境と対応した仮想モデルを用いて、設計段階から運用までを通じた評価・検証を行うことが重要となる。

本検討では、データセンターの構造、電力消費、冷却挙動、運用状態を統合的にモデル化し、システム構成や運用条件の変化に対する影響を事前に評価可能な環境の構築を目指す。これにより、ラック配置、冷却水流量制御、電力制約下での運用方針、設備冗長構成などについて、定量的な評価に基づく設計判断を可能とする。

デジタルツイン基盤としては、オープンソースで開発が進められている ExaDigiTを活用することを想定している。本基盤は、HPCシステムおよびデータセンターの運用状態を再現・分析するためのフレームワークであり、消費電力、熱挙動、ワークロード実行状態などを統合的に扱うことが可能である。これにより、計算機と建屋設備を含めたシステム全体の挙動をモデル化し、設計および運用最適化に資する評価基盤として活用する。

今年度においては、本デジタルツイン基盤の有効性を検証するため、計算科学研究センターの冷却施設や今後導入予定のAI4Sスーパーコンピュータを対象にモデル構築および評価を実施した。詳細設計においては、富岳NEXTの計算機や冷却設備モデルを構築し、建屋設計およびシステム設計のコデザインを高度化するとともに、運用段階においても継続的な最適化および異常検知への活用を図る予定である。将来的には、運用データ基盤と連携することで、実システムの状態をリアルタイムに反映した動的なデジタルツインへと発展させることを目指す。

3.5.7 建屋の検討状況

本節では、「富岳NEXT」の設置を前提とした建屋に関する検討状況について示す。

まず、計算サービスの継続性の観点から、「京」から「富岳」への移行時のように既存システムを撤去した後に同一敷地へ新システムを設置する方式ではなく、「富岳」と「富岳NEXT」を一定期間並行して稼働可能とする構成を採用することとした。このため、既存建屋を利用するのではなく、新たに建屋を整備し、システム移行期間中におけるサービス継続性および利用者環境の安定性を確保する方針とする。

建屋は単なる計算機の設置空間ではなく、電力供給、冷却、保守運用、環境負荷の観点において、計算機システムと一体として設計されるべき基盤である。本プロジェクトでは、フィージビリティスタディにおける検討をもとに、設計・建設・運用を一体として最適化するDBO（Design-Build-Operate）方式を前提として検討を進めている。本方式により、建屋インフラと計算機システムとの整合性を確保するとともに、設計段階から運用効率およびライフサイクルコストの最適化を図る。

建屋設計において特に重視するのは、エネルギー効率の飛躍的な向上である。従来の国内データセンターでは、IT電力に対して冷却電力が約30～50%を占める構成が一般的であったのに対し、富岳NEXTでは冷却電力比率を大幅に低減し、世界最高水準に近い効率の実現を目標とする。具体的には、「富岳」において約35%であった冷却電力割合を、10%程度まで低減することを設計目標とする。この実現に向け、チラーに依存しないフリークーリングを前提とした温水冷却方式を採用し、エネルギー効率の向上と環境負荷低減を両立する。

建屋整備事業は、2026年2月の募集要項公表を起点として開始され、競争的対話方式による事業者選定プロセスを経て、2026年内に基本契約締結に至る計画である。2026年3月から6月にかけて複数回の競争的対話および質疑応答を実施し、技術的要件および事業条件の精緻化を図る。その後、同年8月に提案書の受付、9月にプレゼンテーションおよびヒアリングを実施し、10月に優先交渉権者を選定する予定である。選定後、基本協定および基本契約を締結し、事業を次段階へと進める。

基本契約締結後は、本施設の設計・施工フェーズに移行し、2029年12月の施設引渡しを期限として建屋および関連インフラの整備を実施する。この期間においては、計算機システム仕様の確定と並行して、電力設備、冷却設備、建屋構造、レイアウト設計を統合的に具体化し、施工および設備据付を段階的に進める。

施設引渡し後は、2036年3月までの維持管理フェーズに移行する。本フェーズでは、建屋設備の安定運用および保守管理を実施するとともに、計算機システムの運用と連携した継続的な最適化を行う。これにより、長期にわたり高効率かつ安定した計算機の運用を維持することを目指す。

以上のように、本建屋は単なる施設整備にとどまらず、「高効率・低環境負荷・高密度収容」を実現する次世代データセンターモデルとして位置づけられるものである。今後の詳細設計においては、システム仕様および運用要件との整合を図りつつ、電力・冷却・設置条件を含めた最適化を継続的に進める。

3.6運用設計

本節では、富岳NEXTの運用システムおよび利用環境整備に関する基本設計段階での検討状況を報告する。本節は、次世代フラッグシップシステムにおける運用基盤の全体像を示すとともに、その設計思想、検討の前提条件、および詳細設計へ引き渡すべき基本方針を整理することを目的とする。富岳NEXTは、従来の高性能計算基盤の延長線上にあるシステムではなく、AI・シミュレーション・データ解析が混在する新しい利用形態を前提とした計算基盤として設計される。

スーパーコンピュータ「富岳」は、我が国の基幹的HPCシステムとして、科学技術計算から産業応用、社会課題解決に至るまで多様な分野を支えてきた。大規模数値シミュレーションを中心とした従来型のHPCワークロードにおいては、高い安定性と信頼性を実現し、世界的にも高い評価を得ている。一方で、近年の研究環境の変化に伴い、HPCを取り巻く要求は大きく変容している。AIおよび機械学習の急速な発展、データ駆動型研究の拡大、外部データ基盤やクラウドとの連携需要の増大、さらには対話的利用やAPI主導型利用の増加などにより、従来のバッチ中心型運用を前提としたシステム構造では対応が困難となる場面が増えている。

富岳の運用経験を通じて、いくつかの構造的課題が顕在化した。ワークロードの多様化に対する柔軟性の不足、利用者インターフェースの高度化要求への対応、外部システムとのデータ連携の複雑化、セキュリティ要件の高度化、さらにはシステム変更時の影響範囲の大きさなどが挙げられる。これらの課題は、単なる機能追加や部分的な改修では解決できず、運用システム全体の設計思想を再検討する必要があることを示している。

「富岳NEXT」は、GPUを中核とした計算基盤を採用し、AI・シミュレーション・データ解析を統合的に扱う中核計算基盤となることが求められる。このような環境においては、計算資源の公平かつ高効率な管理、多様な利用形態の並存、外部システムとの高速かつ安全なデータ連携、セキュリティ設計、ならびに可観測性と自動化を備えた運用基盤の整備が不可欠である。さらに、2030年に向けてはAI駆動型運用の重要性が増すことが想定されるため、短期的な最適化ではなく、持続可能性と拡張性を備えた運用システム設計が必要となる。

本報告書が対象とする範囲は、富岳NEXTにおける運用管理システムの基本方針、利用者向け利用環境の基本方針、認証・認可およびアカウント管理の考え方、外部システムとの連携方針、ならびに可用性・運用性・拡張性・セキュリティといった非機能要件の整理である。これらはシステムの運用を担保するための基礎的な事項であり、基本設計段階で方向性を確定しておく必要がある。一方、個別技術や製品の選定、具体的な構成設計、詳細な実装方式やパラメータ設計、運用手順書や作業フローの記述については、本報告書の対象外とし、詳細設計以降の工程に委ねる。

本報告書は、まず、現行システムである富岳における運用・利用環境上の課題を整理するとともに、2030年を見据えた要求事項を分析する。その上で、富岳NEXTに求められる運用基盤の全体像を示し、サービスモデルや論理アーキテクチャ、可用性・柔軟性・拡張性といった設計観点をまとめる。続いて、ワークロード管理、システムソフトウェア、ユーザサービス、運用支援機能、ネットワーク連携、運用プロセスなど、個別コンポーネントごとに基本設計方針を整理する。それぞれの節では、基本設計として確定すべき事項と、詳細設計以降で検討すべき項目を明確に区分し、後続フェーズへ引き継ぐための前提条件および設計思想を示している。

3.6.1 現状分析と課題・要求事項

3.6.1.1 富岳からの課題

本節では、「富岳」の運用開始から2025年9月時点までを対象とし、主として運用保守メンバーへのヒアリングおよび運用実績の分析に基づき抽出された課題を整理する。抽出した課題を個別事象として扱うのではなく、その背後にある構造的要因を明確化し、根本原因として分類する。その上で、各根本原因に対して「富岳NEXT」における設計方針および要求事項を導出することを目的とする。

本分析は、単なる反省事項の列挙を目的とするものではない。将来的な利用形態の変化や技術進展を前提とし、持続可能かつ拡張可能な運用基盤を構築するための設計指針を確立することを最終目的とする。すなわち、「富岳」で顕在化した課題を再発防止の観点から体系化し、「富岳NEXT」における設計上の拘束条件へと転換することを目指す。

3.6.1.1.1　運用フェーズにおける機能開発不足と設計思想の乖離

「富岳」は運用開始以降、安定性および可用性を最優先とした保守運用が継続されてきた。その結果、運用フェーズにおいて大規模な機能改修や設計思想の再整理は実施されず、主として小規模な機能追加や部分的な拡張によって対応してきた経緯がある。このような漸進的な改修の積み重ねは短期的な安定性確保には寄与したものの、長期的には以下の構造的課題を生じさせた。

第一に、当初設計思想との乖離である。「富岳」は大規模バッチ処理を中心とする利用形態を前提に設計されており、AI解析やデータ駆動型処理、対話型利用といった新たなワークロード形態は想定の中心にはなかった。そのため、近年の利用要求の多様化に対して、設計思想そのものが十分に整合していない部分が顕在化した。

第二に、機能拡張の非体系化である。個別課題への対症療法的な改修が重ねられた結果、システム全体としての統一性および整合性が徐々に低下した。これにより、構成の複雑化、依存関係の増大、保守作業の属人化などが発生し、開発・保守コストの増加や利用者体験の不整合につながった。

以上より、「富岳NEXT」では、運用開始後の改修を前提とした設計思想を明確化することが要求される。すなわち、当初設計と運用後との乖離が生じることを前提として、拡張性・変更容易性を設計段階から組み込む必要がある。

3.6.1.1.2　設計時の運用仕様不足とそれに伴う課題

「富岳」の設計段階においては、主として機能仕様および性能要件が中心に議論され、運用仕様の体系的定義は十分とは言えなかった。その結果、運用フェーズにおいて管理体制や運用手順が逐次整備される形となり、設計と運用の間に断絶が生じた。このことは、以下のような課題として現れている。

第一に、定型業務の自動化不足である。多くの運用作業が手動または半自動で実施されており、自動化による効率化および人的ミス低減の余地が存在する。特に大規模ノード更新や設定変更作業においては、作業時間の長期化および運用負荷増大が課題となった。

第二に、情報管理の分散である。運用情報、構成情報、監視データなどが複数系統で管理されており、情報の一元性および整合性確保が課題となっている。このことは、障害解析や変更管理の迅速性にも影響を及ぼす。

第三に、可観測性の不足である。システム状態を包括的に把握するための監視基盤は整備されているものの、運用最適化や将来的な高度分析を前提としたデータ設計は十分とは言えない。運用の安定性向上および利用者への情報提供充実の観点から、監視および可観測性の設計は再整理が必要である。

以上を踏まえ、「富岳NEXT」では設計段階から運用仕様を明示的に定義し、運用プロセスを含めた設計とすることが求められる。自動化を前提とした構成設計、情報の一元管理、可観測性の強化を基本要求事項として整理する。

3.6.1.1.3　HPC ミドルウェア依存による制約

「富岳」においては、HPCミドルウェア(富士通製TCS)への依存度が高く、システム制御およびジョブ管理の多くが当該ミドルウェアに依拠している。この構成は、初期段階では効率的な運用を可能にした一方で、長期的には以下の制約を生じさせた。

第一に、更新柔軟性の制約である。特定ミドルウェアへの強い依存は、OS更新や周辺機能改修時の影響範

囲を拡大させ、継続的な改善を困難にする要因となった。

第二に、最新技術への対応遅延である。AI基盤やクラウド連携機能など、近年急速に進展している技術領域との整合性確保において、既存構成が制約条件となる場面が見られた。

第三に、標準化との乖離である。現代のセキュリティ要件や外部システム連携においては、OSSベースの標準的プロトコルやインターフェイスとの整合性が重要となるが、独自性の高い構成はその柔軟性を制限する可能性がある。

以上を踏まえ、「富岳NEXT」では特定ミドルウェアへの過度な依存を回避し、標準化およびOSS活用を基本方針とすることが要求される。これにより、継続的更新、他基盤との相互運用性確保、セキュリティ強化への柔軟な対応を可能とする。

3.6.1.2 要求事項

「富岳NEXT」は、従来の大規模MPI中心の計算環境から、AI・データ解析・対話型処理・常駐サービスを含む多様なワークロードが混在する統合計算基盤へと進化することが前提となる。そのため、本節では2030年時点で想定される利用形態および技術環境を踏まえ、各構成要素に対して要求される機能・設計原則を整理する。

本節で示す要求事項は、個別技術の選定を規定するものではなく、基本設計段階で確定すべき要件を定義するものである。

3.6.1.2.1 ワークロード管理

「富岳NEXT」が扱うワークロードは、従来型の大規模MPIジョブに加え、AI/ML処理、データ解析、イベント駆動型処理、対話型処理、常駐型サービスなど、性質の異なる処理が同一基盤上で同時に実行されることを前提とする。これにより、スケジューリング、リソース割当、キュー設計、実行環境制御および利用状況可視化といった運用要素は、従来以上に複雑な状態を前提として設計する必要がある。

特に、GPU集中型ノードおよびヘテロジニアス構成の一般化により、CPU、GPU、メモリ、ストレージ帯域、ネットワーク帯域など複数の資源を統合的に管理し、ジョブ特性に応じて最適な割当方式を選択可能とする仕組みが不可欠となる。また、省エネルギー要件、クラウドバースティング、フェデレーション利用など、外部基盤との連携を前提とした資源制御も考慮する必要がある。

以上を踏まえ、「富岳NEXT」のワークロード管理は、従来型HPC向けWLM機能とクラウドネイティブ基盤の管理機能を統合した柔軟な管理基盤として構築されなければならない。

3.6.1.2.2 システムソフトウェア

システムソフトウェアには、従来型HPCアプリケーションへの対応に加え、AI・データ駆動型処理・クラウド連携・フェデレーション利用など、多様化する研究形態を単一基盤上で統合的に扱うための柔軟性および持続可能性が求められる。特に、GPUの一般化に伴い、OS、コンパイラ、ライブラリ、MPI、I/Oスタック等のソフトウェア階層は、高度な性能最適化を実現すると同時に、継続的な更新および拡張が可能な構造として維持されなければならない。性能最適化と更新容易性を両立する体系的設計が不可欠である。

アプリケーション実行環境については、従来のモジュール管理方式に加え、コンテナなど複数の実行形態を前提とする構成とし、再現性および可搬性を確保しながら安全な隔離を実現する仕組みを備える必要がある。特にAIワークロードにおいてはライブラリ更新頻度が高く、依存関係が複雑化する傾向にあることから、手動管理に依存した運用は持続困難である。依存関係解析、バージョン管理、脆弱性検知を含む統合的な管理機構の整備が求められる。

さらに、国際HPCコミュニティとの互換性を維持しつつ、ベンダー最適化技術を継続的に取り込むことができる構造とすることが重要である。標準APIおよび共通インターフェースを整備することで、アプリケーション開発者、利用者、

運用者が共通基盤上で効率的に作業可能な環境を構築する。

以上を踏まえ、システムソフトウェアは「富岳NEXT」の運用基盤を安定的に支えるとともに、2030年の多様な計算需要に適応可能な、柔軟かつ持続的なソフトウェア体系として整備されなければならない。

3.6.1.2.3　ユーザサービス

利用者向けのインターフェイスとして、多様な利用者が専門知識の有無にかかわらず、高度なHPC基盤を効率的に利用できることを目的としインターフェイスを整備する。従来のCLI主体の操作体系に加え、Webポータル、GUIアプリケーション、API連携など多様な利用形態を単一のユーザサービス層として統合し、利用者のスキル差を吸収しながら一貫性のある操作環境を提供する仕組みが求められる。

対話型利用や可視化、AI・データ解析用途の拡大に伴い、ログイン環境、対話型実行環境、可視化環境、データ転送機能など、従来はフロントエンドシステムとして個別に扱われてきた機能についても、ユーザサービスの一部として統合的に提供する必要がある。これにより、利用者が実行形態や基盤構成の違いを意識することなく、目的に応じた計算、解析、可視化およびデータ操作を一貫した操作体系で実施できる環境を実現する。

さらに、多言語対応、利用状況の可視化、操作体系の統一、アカウント連携機能など、国際的研究基盤としての利用者体験を重視した設計が不可欠である。ユーザサービスは、ワークロード管理、運用支援システム、運用データ基盤と密接に連携し、利用者がジョブ、データ、ワークフローを統合的かつ直感的に扱える構造としなければならない。

以上を踏まえ、「富岳NEXT」のユーザサービスは、従来型HPC利用者およびAI・データ科学分野の利用者の双方にとって最適な入口となる統合UX基盤として整備する。

3.6.1.2.4　運用支援システム

運用支援システムは、従来のアカウント管理、課題管理、課金処理といった基盤業務の効率化にとどまらず、HPCを用いた研究活動全体を支える統合的な支援基盤として機能することが求められる。利用形態の多様化および利用者数の増加に伴い、運用業務は高度化・複雑化している。個別システムや手作業に依存した運用体制では、効率性、透明性および整合性の確保が困難となる。

そのため、利用者情報、課題情報、ジョブ実績、課金・配分情報、問い合わせ履歴等を、統合データモデルに基づき一元管理することが重要である。これにより、運用効率の向上、処理の透明性確保、およびデータ整合性の維持を実現する。

さらに、外部連携やフェデレーション利用の一般化を踏まえ、HPCIとの連携、外部システムとのデータ連携、セキュリティ要件に基づくユーザ属性管理等に対応可能な拡張性を備える必要がある。将来的な制度変更や利用形態の変化にも柔軟に適応可能な設計とすることが求められる。

また、運用支援システムは運用データ基盤と密接に連携し、利用状況データ、運用情報、監査情報を統合的に扱うことで、運用状況の可視化および高度化を支援する基盤として機能しなければならない。

以上を踏まえ、運用支援システムはユーザ・課題・運用情報管理基盤として位置づけ、「富岳NEXT」の全体運用を効率的かつ安全に支える中核コンポーネントとして整備する。

3.6.1.2.5　運用基盤

運用基盤は、数千ノード規模の大規模システムを長期間にわたり安定的に運用することを前提として、安定性、可観測性および自動化の高度化を中核に設計されなければならない。従来の個別スクリプトに依存した運用や、部分的に分散した監視・管理基盤では、ヘテロジニアス構成の進展、GPU集約型ノードの増加、AIワークロードの拡大に伴う負荷変動、ならびにセキュリティ要件の高度化といった複雑化した運用環境に対応することは困難であ

る。

このため、運用基盤は、プロビジョニング、構成管理、監視、ログおよびメトリクス収集、イベント管理、障害対応、利用者情報管理、ジョブ情報管理などの各要素を統合的に扱う基盤として構築する必要がある。Infrastructure as Code（IaC）およびObservabilityを基本原則とし、構成変更、状態把握、障害対応を人手に過度に依存せず実行可能な設計とすることが不可欠である。

ジョブ情報を含む運用ログ、性能メトリクス、イベント情報、監査情報を一元的に収集・蓄積・分析することで、運用状態の可視化、性能最適化、障害予兆検知を実現し、自律的な運用高度化を可能とする基盤を整備する必要がある。

さらに、計算ノードの用途がバッチ処理、対話型利用、常駐サービスなど多様化することを前提とし、クラスタ構成変更、ノード増設、運用ポリシー変更等に迅速に追従可能な柔軟性を備える必要がある。管理系、制御系、計算系の役割を明確に分離し、障害発生時の影響範囲を局所化することで、システム全体の信頼性を確保する設計とする。

以上を踏まえ、運用基盤は「富岳NEXT」全体の信頼性、効率性および拡張性を支える統合的かつ自律的な運用コアとして整備されなければならない。

3.6.1.2.6　セキュリティ

HPCは従来、閉域環境における安定動作を前提として設計・運用されてきた。しかし「富岳NEXT」世代においては、外部クラウド、観測装置、産業システム、HPCIフェデレーション、海外研究機関等との接続が一般化することが想定される。このような環境では、従来の境界型防御モデルのみでは十分ではなく、通信およびアクセス制御を前提とした包括的なセキュリティ設計が必要となる。

そのため、認証および認可機構の強化、ネットワークの明確な分離、監査ログの統合管理などを組み合わせた多層防御構造を基本とする。一方で、高速データ転送要件と、外部接続に伴うセキュリティ要件を両立させる設計が求められる。高帯域通信を確保しつつ、制御系、管理系、計算系のネットワークを論理的および物理的に分離し、用途に応じた防御レベルを適用可能とする構造とする。

多様な利用形態に統一的に対応しながら性能と安全性を両立するため、「富岳NEXT」では、通信可視化、異常検知、ログ管理、分離アーキテクチャを組み合わせたセキュリティ設計を基本方針とする。セキュリティは独立した単機能ではなく、運用基盤と連携した継続的かつ運用主導型の仕組みとして設計されなければならない。

3.6.1.2.7　運用プロセス

運用プロセスは、従来のHPC運用において一般的であった手作業中心の保守、監視および変更管理から進化し、自動化、標準化、継続的改善、ならびにAIOps連携を基本とする運用様式へ移行する必要がある。ワークロードの複雑化、頻繁なライブラリアップデート、クラウド連携、外部機関とのフェデレーション利用などにより、運用負荷は従来のHPC環境と比較して大幅に増大することが想定される。このような環境においては、属人的な対応や非体系的な手順では安定運用を維持することは困難である。

そのため、手順の明確化および標準化（SOPの整備）、変更管理の体系化、インシデント対応の迅速化、予兆保守の導入、サービスレベル目標（SLA/SLO）の明確化、ならびに文書管理の一元化を実施する必要がある。加えて、利用者層の多様化により、教育機関、企業、国際共同研究者等が混在する環境において、公平性および透明性を確保した運用プロセスを維持することが求められる。また、AI駆動型運用を実現することも求められる。

運用プロセスは、単なる手順書の整備にとどまらず、運用データの活用による継続的改善サイクルを内包した体系として設計されなければならない。これにより、システム規模の拡大や利用形態の変化に対しても持続的に対応可能な運用基盤を確立する。

3.6.1.2.8　導入・運用計画

導入および運用計画は、単なる新規システム立ち上げの枠組みを超え、複雑化・高度化する研究基盤を持続的に支えるライフサイクル計画として策定されなければならない。導入と運用は分離された工程ではなく、一貫した設計思想のもとで統合的に管理される必要がある。

まず導入フェーズにおいては、建屋整備、電力・冷却設備構築、ネットワーク設計、ハードウェア設置に加え、ユーザサービス、ワークロード管理（WLM）、仮想化基盤、運用データ基盤、運用支援システムなど、全サービス層を横断した計画を策定する必要がある。ハードウェアの物理設置のみをもって導入完了とするのではなく、ソフトウェアスタック、管理基盤、ユーザ体験層までを含めた一体的な立ち上げを前提とする。

特に「富岳」から「富岳NEXT」への移行においては、既存のユーザ環境、アプリケーション資産、データ資産の継続利用を前提とした設計が不可欠である。互換性の確保、移行支援体制の整備、並行運用期間の設定、段階的フェーズ導入を計画的に実施することにより、利用者への影響を最小限に抑える必要がある。また、性能検証、スケジューリング挙動の確認、AI/MLワークロード動作検証など多面的な検証を体系的に実施する体制を導入計画に組み込む必要がある。導入初期フェーズにおける安定性確保、利用者教育、ドキュメント整備も、導入工程の重要な構成要素と位置付ける。

導入後の長期運用フェーズにおいては、5年以上の安定稼働を前提としつつ、技術進化への継続的対応を可能とする構造を維持することが重要である。ソフトウェアスタックおよびライブラリ更新の頻度が高くなることが想定されるため、計画的な更新戦略と拡張性を備えた設計を維持する必要がある。HPCI連携、国際共同研究、クラウド活用といった外部基盤との接続を前提に、標準化および互換性を重視した構成を継続的に保つことも求められる。

さらに、運用体制の持続性も重要な要素である。運用自動化の推進により属人化を抑制し、知識の体系化および継承を可能とする体制を構築する必要がある。また、災害リスクや電力制約といった外部要因を考慮し包括的なリスク管理体制を整備することが求められる。

以上を踏まえ、導入および長期運用計画は、持続性、進化性、更新容易性を基本軸とし、「富岳」からの利用者移行を確実に実施するとともに、将来的な拡張および制度変更にも柔軟に対応可能な戦略的計画として策定されなければならない。

3.6.1.2.9　運用の AI 化

「富岳NEXT」においては、AIを単なる利用者向け計算資源として提供するだけでなく、運用そのものを高度化する基盤技術として位置づける。従来のHPC運用では、障害対応、性能分析、ログ解析、問い合わせ対応、設定変更、ソフトウェア更新など、多くの業務が運用担当者の経験と手作業に依存してきた。しかし、ワークロードの多様化、システム構成の複雑化、ソフトウェア更新頻度の上昇に伴い、これらを人手のみで継続的かつ高品質に実施することは困難になりつつある。そのため、「富岳NEXT」では、運用業務の実行、改善、およびそれを支えるソフトウェア開発の各段階にAIを組み込むことを基本方針とする。

まず、運用実務の面では、ログ、メトリクス、イベント、ジョブ情報、問い合わせ履歴、構成情報などをもとに、AIが障害兆候の検知、原因推定、対応候補の提示、問い合わせ応答、設定変更支援、運用判断支援を行うことを前提とする。これにより、従来は運用担当者が個別に実施していた調査や判断を支援し、対応の迅速化、品質の均一化、属人性の低減を図る。さらに、一定の条件下では、ノード隔離、サービス再起動、ジョブ再投入、通知発行などの定型的な処置について、AI支援のもとで自動的に実行することも視野に入れる。

次に、運用を支えるソフトウェアの開発においても、AIを積極的に活用する。「富岳NEXT」の運用基盤、運用支援システム、ユーザインターフェイス、監視・分析基盤などは、運用開始後も制度変更、利用形態の変化、新規要件の追加に応じて継続的な改修と機能拡張が必要となる。そのため、要求整理、設計補助、実装、テスト、ドキュメント作成、保守作業に至るまで、ソフトウェアライフサイクル全体でAIを活用し、開発速度と保守性の向上を図る

ことを前提とする。これにより、従来は大きな人的負担を伴っていた運用ソフトウェアの改修を、より短期間かつ柔軟に実施可能とする。

さらに重要なのは、AI活用を一時的な自動化にとどめず、運用開始後も継続的な改善サイクルに組み込むことである。「富岳NEXT」では、運用データ基盤に蓄積される実運用データをもとに、運用プロセスそのものを継続的に見直し、改善することを前提とする。例えば、障害対応履歴、利用者からの問い合わせ内容、ジョブ失敗傾向、性能ばらつき、資源利用状況などを分析し、AIが運用手順の見直し、しきい値の調整、監視ルールの改善、運用フローの再設計を支援する。これにより、運用プロセスは固定的なものではなく、実データに基づいて進化し続けるものとなる。

同様に、運用ソフトウェアについても、運用開始後の実績を踏まえた継続的な機能拡張を行う。利用者要求や運用上の課題をAIが整理・分析し、必要な改修項目や新機能候補を抽出することで、運用ソフトウェアを段階的に高度化していく。すなわち、「富岳NEXT」では、AIを活用して運用を実施するだけでなく、AIを用いて運用プロセスを改善し、さらにAIを用いてその改善を実装するという循環を形成することを目指す。

以上を踏まえ、「富岳NEXT」における運用のAI化とは、運用業務の自動化・高度化、運用ソフトウェア開発の効率化、運用開始後の継続的改善を一体として実現するものである。これにより、運用は固定的な仕組みではなく、システムの利用状況や技術進展に応じて継続的に進化する基盤へと転換される。

3.6.2 運用の全体像

本節では、前節で整理した要求事項を踏まえ、富岳NEXTにおける運用の全体像を示す。本節の目的は、個別機能の詳細を説明することではなく、それらを貫く設計思想を明確化することにある。

富岳NEXTは、従来の大規模MPIジョブを中心とした計算環境から、AI・データ解析・対話型処理・常駐サービスなどが混在する統合計算基盤へと進化することが想定される。このような環境では、ワークロード管理、実行環境、ユーザインターフェイス、運用支援、運用基盤、セキュリティ、運用プロセスといった各要素も多様な計算を考慮した設計が必要であり、システム全体として整合の取れた運用設計が不可欠となる。

本運用において中核となるのは、異種ワークロードの統合的な管理である。富岳NEXTでは、従来型のバッチジョブに加え、AI/ML処理、データ解析、対話型処理、常駐型サービスが同一基盤上で同時に実行される。このため、CPU、GPU、メモリ、ストレージ帯域、ネットワーク帯域といった複数の資源を統合的に扱い、ジョブ特性に応じた最適な割当を行う必要がある。また、HPC向けワークロード管理機能とクラウドネイティブワークロードの管理機能を組み合わせたワークロード管理機構により、多様な実行形態を共存させつつ、資源利用効率と柔軟性を両立することが求められる。

これらのワークロードを支える実行環境およびシステムソフトウェアについては、性能最適化と継続的更新を両立する体系として設計する必要がある。特にAIワークロードではライブラリの更新頻度が高く、依存関係も複雑化しやすいため、従来の手動管理に依存した運用は持続困難である。このため、モジュール管理とコンテナを組み合わせた実行環境を前提とし、依存関係管理、バージョン管理、脆弱性対応を含む統合的な管理機構を整備する。

また、運用設計はシステム構成そのものではなく、利用者視点を中心として構築する。富岳NEXTでは、CLIに加え、WebポータルやAPIといった複数の利用形態を単一のユーザサービス層として統合し、利用者が基盤構成や実行形態の違いを意識することなく計算や解析を実施できる環境を提供する。

運用支援システムは、単なる管理機能ではなく、研究活動全体を支える基盤として位置づける。利用者情報、課題情報、資源利用情報、ジョブ性能、問い合わせ履歴等を統合的に管理することで、運用効率と透明性を高めるとともに、利用者の研究活動を支える基盤として整備する。また、これらの情報は運用データ基盤と連携し、運用状況の可視化および分析に活用される。

運用基盤は、可観測性と自動化を中核として設計する。ログ、メトリクス、イベント、監査情報を統合的に収集・蓄積し、システム状態を多角的に把握可能とすることで、障害の早期検知や性能最適化を実現する。また、構成管理およびプロビジョニングを Infrastructure as Code により管理し、構成変更や障害対応を人手に過度に依存しない運用を実現する。これにより、大規模かつ複雑な環境においても、運用コストを抑えつつ持続的な運用を可能とする。

セキュリティについては、ネットワーク分離、テナント分離、認証・認可の統合、監査ログの一元管理などを組み合わせ、従来HPCにおいて軽視されがちであったセキュリティ要件を設計段階から体系的に考慮する。また、外部クラウドや他機関との接続・連携を前提としつつ、安全性と利便性を両立するサービス提供およびアクセス制御を適用する。

運用プロセスは、手作業中心の運用から脱却し、自動化、標準化、継続的改善を基本とする体系へ移行する。変更管理、インシデント対応、問題管理を体系化するとともに、運用データに基づく改善サイクルを組み込むことで、運用品質の持続的向上を図る。また、利用者層の多様化に対応し、公平性および透明性を確保した運用を実現する。

さらに、導入および運用は単一の工程としてではなく、ライフサイクルとして統合的に設計する。「富岳」からの移行を前提とし、既存資産の継続利用、段階的導入、並行運用を計画的に実施する必要がある。導入後においても、長期運用を前提とした更新戦略および拡張性を維持し、技術進展や利用形態の変化に対応可能な基盤とする。

以上のように、富岳NEXTの運用は、異種ワークロードの統合、持続可能なソフトウェア体系、利用者中心設計、データ駆動型運用、自動化、セキュリティ統合、継続的改善を中核とする統合運用基盤として構築される。この運用は、従来の安定運用を目的としたHPCから、研究活動を支援し、継続的に進化する基盤への転換を実現するものである。

3.6.3 システムソフトウェア

3.6.3.1 目的と位置づけ

本節は、富岳NEXTの計算ノードにおけるOSおよび仮想化基盤の採用方針を、詳細設計1の前提条件として定めるものである。対象は計算ノードに限定し、管理ノード等の運用基盤構成については後の章に委ねる。計算ノードの前提が不明確な場合、ジョブ実行方式、ソフトウェア供給、監査要件の解釈が章ごとに不整合を生じる可能性がある。そのため本章では、方式候補の比較ではなく、後工程における設計判断の迷いを排除するための拘束条件を優先して定義する。また、計算ノードは1ノード内で複数のユーザジョブが同時に実行されることを前提とし、マルチテナント成立条件を本章で明確にする。

3.6.3.1.1 スコープ

本節で扱う主要観点は以下のとおりである。

- OS の選定方針
- OS ジッタ低減の設計方針
- FUJITSU-MONAKA-X の OS サポート方針
- 仮想化方式の決定
- コンテナランタイムの選定方針
- 詳細設計 1 で実施する検討項目および成果物

3.6.3.1.2 決定事項

本節の決定事項は、詳細設計1の入力条件として適用する。

- 詳細設計 1 では Ubuntu を初期評価対象 OS とする。
- 計算ノードの仮想化方式はコンテナ方式とする。
- 詳細設計 1 では Apptainer を初期評価対象コンテナランタイムとする。
- OS ジッタ対策は、KPI を定め、測定・分析・改善の運用サイクルに組み込む。
- 本番環境の OS およびコンテナランタイムは、選定方針に従い、システム設置予定日の 1 年前までに確定する。
- 同一ノード内の複数ユーザジョブは、コンテナ単位で実行境界を分離し、プロセス、ファイルシステム、ネットワークの分離を必須とする。
- マルチテナント環境における資源保護として、CPU、メモリ、GPU、ローカルストレージの使用上限をジョブ単位で管理する。

3.6.3.1.3 未決事項

以下は未決事項として残す。

- 本番環境の OS およびコンテナランタイム

3.6.3.2 OS

3.6.3.2.1 ディストリビューション選定方針

OSは以下の条件を満たすことを前提に選定する。選定にあたっては、運用継続性およびセキュリティを重視する。

- GPU、NIC、ストレージ等の各種デバイスを十分にサポートすること

- 長期的な運用継続が可能であること
- セキュリティ更新を継続的に受けられること
- HPC における十分な利用実績を有すること

Linuxディストリビューションを広く調査した結果、上記の条件を満たすディストリビューションとしてUbuntuを有力な候補として選定した。UbuntuはNVIDIAがDGX OSのベースOSとして選定していることからもGPUへの対応が望めること、また長期的なサポートとしてLTSが存在すること、HPCにおける実績としてはRedHat Linuxよりは数が少ないが十分な採用実績を有することが選定の理由である。基本設計の段階では富岳NEXTの計算ノードのOSとして決めるわけではないが、詳細設計1ではUbuntuを評価対象とし運用環境の検証を実施する。

3.6.3.2.2　OS ジッタ対策方針

OSジッタはMPI同期時にノード間の性能差として顕在化するため、設計段階のみならず運用段階においても継続的な対策が必要である。そのため、OSジッタ対策は測定に留まらず、測定結果を運用改善に反映することを前提とした改善プロセスを定義する。測定項目は継続的な比較が可能であることを条件とし、代表的なKPIとして以下を想定する。

- 最大許容レイテンシばらつき
- MPI 同期時の遅延分散
- OS ノイズ発生率

KPIは初期導入時に加え、更新リリースごとに再評価する。評価結果は構成変更や再調整の判断に活用する。具体的な評価条件および閾値、改善のプロセスは詳細設計1で定義する。

3.6.3.2.3　FUJITSU-MONAKA-X 対応

FUJITSU-MONAKA-Xの機能は原則としてLinux等のOSSへ反映し、Linuxディストリビューションにより必要なハードウェアサポートが提供されることを前提とする。基本設計では導入予定のCPU機能について現時点のLinux対応状況を整理した。詳細設計以降において追加開発の要否を検討する。

本調査は基本設計時点で想定されるCPU仕様に基づくものであり、仕様の追加・変更があった場合は、それに応じて対応状況を更新する。対応状況は「対応済み」「一部未対応」「未定」に区分し、「対応済み」は追加開発不要、「一部未対応」は詳細設計において開発要否を判断、「未定」は仕様確定後に再評価するものとする。

FUJITSU-MONAKA-X の Linux 対応状況(2025 年 12 月調査)

CPU の機能	対応状況	詳細
Arm feature 対応 (SVE2, SME2)	対応済み	SVE2 と SME2 の両方とも Linux では既にサポートされている。ただしこれは Linux カーネルとしてのサポートのみであり、各種計算ライブラリ等にはそれぞれの実装方法に基づいて個別に対応が必要である。
電力制御、電力取得	対応済み	電力制御系機能と電力取得系機能のいずれも ACPI に準拠した仕様となっており、基本的に Linux で対応済み。ただし、今後準拠仕様の版数がより新しいものになった場合、対応が必要になる可能性がある。
仮想化機能	一部未対応	ハードウェア仮想化機能（メモリ仮想化、割込み仮想化）、VHE 対応と一部のネスト仮想化に対応済み。ネスト仮想化の内、最新の FEAT_NV3 には未対応。
セキュリテ	一部未対応	ポインタ認証（PAC）と分岐ターゲット識別（BTI）、暗号命令（AES、SHA1、

ィ機能		SHA256、SHA512、SM3、SM4）には対応済み。Confidential Computing に紐づく各種機能（RMEv2、PCIe 暗号化、NVLink-C2C 暗号化）については未対応。
RAS 機能	未定	準拠規格として Arm RAS 及び SBMR が決まっているがその中の詳細な仕様については基本設計では未定の状態であり、Linux の対応状況としても一概に判断できる状況ではない。
パフォーマンスカウンタ	未定	パフォーマンスカウンタについての情報は基本設計では未定の状態であり、Linux の対応状況としても一概に判断できる状況ではない。しかし、パフォーマンスカウンタの特にイベント ID とイベント名の対応情報などは SoC 毎に対応が必要になることが一般的であり、「富岳 NEXT」向けにも必要になる可能性が高い。

3.6.3.3 計算ノードの仮想化

計算ノードの仮想化方式は、性能要件を最優先とし、コンテナ方式を採用する。コンテナ方式は起動効率およびリソース利用効率に優れ、HPCのジョブ実行モデルと整合する。仮想マシン方式は標準実行方式の対象外とし、必要な場合に限り詳細設計1において例外的に検討する。また、1ノードに複数ユーザジョブが共存する前提において、コンテナをテナント境界として運用し、プロセス、ファイルシステム、ネットワークの分離を必須条件とする。

3.6.3.3.1 コンテナランタイムの選定方針

コンテナランタイムは、運用継続性およびセキュリティを満たすものに限定して選定する。選定にあたっては以下の条件を満たすことを原則とする。

- 長期的な運用継続性
- 継続的なセキュリティ更新
- HPC における利用実績

これらの条件を踏まえ、詳細設計1ではApptainerを評価対象とし、運用環境の検証を実施する。

3.6.3.4 詳細設計 1 に向けた評価方針

3.6.3.4.1 検討項目

詳細設計1では、本章で確定済みの前提を入力とし、未決事項を実機相当条件で検証する。

- OS およびコンテナランタイムの継続調査（技術動向を踏まえた再選定を含む）
- Apptainer を前提としたマルチテナント隔離方式と資源管理方式
- OS ジッタ対策の KPI、評価方法、運用プロセスの規定

3.6.3.4.2 成果物

成果物は以下を文章として作成する。

- OS およびコンテナランタイムの継続調査結果
- Apptainer によるマルチテナント実現手順
- OS ジッタ評価方法と改善サイクル案

3.6.4 ワークロード管理

本章は、富岳NEXTにおけるワークロード管理の基本方針を定め、詳細設計以降の方式選定および運用設計を拘束する設計判断基準を規定することを目的とする。富岳NEXTは、同一基盤上でHPC向けワークロードとKubernetesにより管理されるワークロードの双方を提供することを目標とする。そのため、管理方式の役割分担および運用上の責務分界を基本設計段階で明確化する。

本章では、想定するワークロードの種類、候補とするワークロード管理ソフトウェアの位置づけ、ならびに詳細設計1で確定すべき検討項目および成果物を定義する。なお、本章ではHPC向けワークロードを管理する従来型のジョブ管理ソフトウェアをジョブスケジューラと呼ぶ。

3.6.4.1 目的と位置づけ

3.6.4.1.1 スコープ

本章では、ワークロード管理をジョブスケジューラとKubernetesの二系統として整理し、それぞれが担う領域を設計方針として規定する。あわせて、ジョブスケジューラの候補範囲およびKubernetesの適用範囲を明示し、2系統併用を前提とした詳細設計1における検討項目と成果物を示す。本章の対象外は以下のとおりとする。

- 各ソフトウェアの優劣評価および最終的な採否の決定
- 実装方式、設定例、具体的な設定値、構成図の詳細
- 性能チューニング手法および運用手順の確定
- 個別製品の最終確定

3.6.4.1.2 決定事項

富岳NEXTにおけるワークロード管理に関する基本構成として、従来型HPC向けジョブスケジューラとKubernetesの二系統を併用する運用とする。これに基づき、役割分担は以下の方針とする。

- 大規模並列計算、バッチ実行および対話型利用を中心とする HPC 向けワークロードは、ジョブスケジューラで管理することを原則とする
- AI 学習・推論およびサービス型ワークロードは、Kubernetes で管理することを原則とする
- 役割分担の境界条件および例外については、詳細設計 1 において定義する。また、2 系統併用を運用として成立させるため、以下の拘束条件を設ける。
- 各系統で取得される資源利用情報は、CPU 時間、GPU 時間、メモリ使用量、実行時間といった共通計測項目で統一し、課金および監査の観点で整合性を確保する
- ノード内の資源分割は、事前定義した資源プロファイル単位で提供し、標準運用では過剰割当を許容しない

3.6.4.1.3 未決事項

以下の事項については基本設計では確定せず、詳細設計において検討・確定する。

- ジョブスケジューラの製品名

3.6.4.2 想定するワークロードの種類

富岳NEXTでは、AI処理およびサービス型ワークロードへの対応を前提とし、以下の二種類のワークロードが同一基盤上に共存する構成を採用する。従来はこれらを別システムで提供することが一般的であったが、富岳NEXTで

は統合基盤として提供する。

3.6.4.2.1　HPC 向けワークロード

HPC向けワークロードは、大規模並列計算を中心とし、バッチジョブおよび対話型ジョブを含む。多数ノードを同時に確保し、同期を伴う並列実行を前提とするため、公平性および資源利用効率を重視した運用が求められる。

主な対象は以下とする。

- 大規模並列計算
- バッチジョブ
- インタラクティブジョブ

3.6.4.2.2　Kubernetes で管理されるワークロード

Kubernetesで管理されるワークロードは、AI学習・推論およびサービス型ワークロードを想定する。短時間ジョブの混在、需要変動への追従、常駐サービスの継続提供を前提とし、HPC向けワークロードとは異なる運用要件を持つ。

主な対象は以下とする。

- AI 学習・推論ジョブ
- サービス型ワークロード

AI学習・推論ジョブであってもジョブスケジューラを使う場合も多く、必ずしも利用者にKubernetesの利用を強いるものではない。Kubernetesで管理されるワークロードについては、具体的な利用者のユースケースを調査し、需要を想定した上で検討する必要がある。基本設計においては、ユースケースは未調査であり詳細設計において、AIソフトウェアWGと協力し、ユースケースの抽出を実施する。

3.6.4.3　従来型 HPC 向けワークロード管理ソフトウェアの選定方針

富岳NEXTでは、ジョブスケジューラ候補を以下の観点で抽出する。

- 大規模 HPC 環境における運用実績
- 継続的な開発体制および保守性
- 二系統併用および資源分割運用への適合性

基本設計におけるジョブスケジューラの候補は以下の三つとする。

- Slurm
- PBS Pro
- Flux

ジョブスケジューラはHPC向けワークロード運用の中核機能として位置づけ、採否および適用条件は詳細設計1で評価する。評価にあたっては、富岳の運用実績および運用規模を参考とし、運用成立性と柔軟性を主要な判断軸とする。

3.6.4.4　Kubernetes 型ワークロード管理の位置づけ

富岳NEXTにおけるKubernetesは、従来型HPCジョブスケジューラの代替としてではなく、AIおよびサービス型ワークロードに適した領域を担う基盤として位置づける。適用範囲、対象ユースケース、運用前提条件については、詳細設計1において抽出・評価し、確定する。

3.6.4.5 詳細設計 1 に向けた評価方針

詳細設計1では、ジョブスケジューラ候補およびKubernetesを実環境に近い条件で評価し、その結果に基づき採用方針および適用範囲を確定する。評価は、二系統併用による運用成立性を共通の判断軸とする。

3.6.4.5.1 前提条件

評価における前提条件は以下のとおりとする。

- セキュリティ確保の観点から、ジョブはコンテナを起点として実行する
- ノード資源は CPU、メモリ、GPU 単位で分割し、利用者に提供可能であること
- ジョブスケジューラおよび Kubernetes は、標準 OS およびコンテナランタイムの選定方針に従うこと

3.6.4.5.2 検討項目

評価では、以下の観点を中心に検討を行う。

- ジョブスケジューラの性能および運用柔軟性の評価
- 課金連携の実現性
- Kubernetes のユースケース抽出および実現性評価
- 二系統併用における資源分離方式の適合性
- 大規模ジョブにおけるステージング・データ配布方式の実装および運用負荷

3.6.4.5.3 成果物

成果物は、採用判断および運用移行に直接利用可能な形で文章として整理する。

- ジョブスケジューラの評価結果
- Kubernetes の適用範囲、前提条件およびユースケース
- 課金連携に関する調査・評価結果
- 資源分離方式の検討結果
- 大規模ジョブのステージング・データ配布方式に関する検討結果

3.6.5 ユーザ向けソフトウェアスタック

本節では、「富岳NEXT」におけるライブラリ層以上のソフトウェアスタックについて、個別ソフトウェアの選定結果を示すものではなく、運用開始後の長期的な利用および継続的な拡張、多様な利用形態に対応するための標準実行環境の提供方針と、その維持・更新の考え方を基本設計として定義する。本基本設計では、ライブラリ層以上のソフトウェアスタックについて、何を「標準」として提供するか、どのような条件を満たすことを要求するか、どこまでを基本設計で決定し、どこからを詳細設計に委ねるかを明確にすることを目的とする。

3.6.5.1 標準ソフトウェアスタックの提供モデル

「富岳NEXT」におけるライブラリ層以上のソフトウェアスタックは、利用者が研究開発を継続的かつ効率的に実施できる標準実行環境として提供することを基本方針とする。標準スタックは、個別ソフトウェアの寄せ集めではなく、相互に整合性の取れた統合環境として構成されることを前提とする。

標準スタックは、以下の条件を満たすことを要求する。

- 国際的な HPC ソフトウェアコミュニティにおいて広く利用され、継続的な開発が行われていること
- 「富岳」およびバーチャル富岳、関連環境における利用実績を有し、既存のアプリケーション資産との連続性を確保できること
- 特定ベンダーに依存せず、長期的な保守および更新が可能であること

これらの条件を満たすことにより、「富岳NEXT」の標準ソフトウェア環境が一過性の構成とならず、持続的に利用可能であることを担保する。

3.6.5.2 提供範囲の整理（標準・推奨・利用者管理）

ライブラリ層以上のソフトウェアについては、運用負荷の抑制および標準環境の肥大化を防ぐ観点から、提供範囲を以下の考え方に基づき整理することを基本方針とする。

- 標準提供ソフトウェア
  「富岳 NEXT」の利用において基盤的な役割を果たす数値計算ライブラリ、並列処理関連ライブラリ、主要なミドルウェア等については、センターが責任を持って整備・提供する標準スタックに含める。
- 推奨ソフトウェア
  分野や利用形態に応じて有用性が高いが、すべての利用者に必須ではないソフトウェアについては、推奨ソフトウェアとして位置づけ、整備・提供の範囲を限定する。
- 利用者管理ソフトウェア
  特定プロジェクトや個別研究に依存するソフトウェアについては、利用者またはプロジェクト側で管理することを前提とし、標準スタックには含めない。

本基本設計では、この区分方針を決定し、具体的なソフトウェアの分類については詳細設計段階で決定する。

3.6.5.3 更新・互換性および再現性に関する基本方針

標準ソフトウェアスタックは、長期運用を前提としつつも、技術進展に対応するため定期的な更新が可能な構成であることを要求する。一方で、研究成果の再現性を確保する観点から、更新による影響を利用者が把握・制御できることを基本条件とする。このため、標準スタックの更新にあたっては、安定性を重視した構成と、脆弱性対応や新しい技術を取り込む構成を併存させることを前提とする。また、過去の計算環境を再現可能な形で保持・利用できる仕組みを備えることを要求する。

具体的な更新頻度、サポート期間、互換性維持の方法については、本基本設計では方針のみを示し、詳細設

計段階で検討・決定する。

3.6.5.4　配布形態および実行環境との関係

ライブラリ層以上のソフトウェアスタックは、再現性および可搬性を確保する観点から、コンテナ技術との連携を前提とする。標準スタックは、コンテナ環境を基本としつつ、必要に応じてモジュール管理およびパッケージ管理を通じて利用できる構成であることを要求する。これにより、オンプレミス環境に加え、クラウドや外部計算資源との連携においても、同一の実行環境を利用可能とする。

3.6.5.5　国際 HPC ソフトウェアコミュニティ・スタックとの整合

「富岳NEXT」のライブラリ層以上のソフトウェアスタック設計においては、E4SやOpenHPCに代表される国際的なHPCソフトウェアスタックの動向を、妥当性を確認するための参考情報として位置づける。これらの取り組みが示す、オープン性、ポータビリティ性、スケーラビリティ性といった共通の方向性は、「富岳NEXT」の基本設計においても重視する。一方で、特定のソフトウェア集合や配布形態に固定されることは避け、本基本設計では原則および前提条件のみを定義する。

3.6.5.6　本節で決定する事項と詳細設計に委ねる事項

本節において、基本設計として決定する事項は以下のとおりである。

- ライブラリ層以上のソフトウェアスタックは、標準実行環境として統合的に提供する
- オープンソースを中心とし、国際的な HPC ソフトウェアスタックとの整合を重視する
- 提供範囲は「標準」「推奨」「利用者管理」に区分する
- 更新可能性と再現性を両立する構成を前提とする

一方、以下の事項については詳細設計段階に委ねる。

- 個別ソフトウェアの採否およびバージョン構成
- 更新頻度および具体的なサポートポリシー
- コンテナ利用やモジュール管理・パッケージ管理の具体的方式および運用手順

3.6.6 ユーザサービス

3.6.6.1 ユーザインターフェイス

従来のCLI中心のシステムから脱却し、これら3インターフェイスが同質的に機能する統合環境の構築を基本方針とし、利用者のスキルレベルや目的に関わらず、一貫した操作体験を提供することを目指す。この方針は、富岳におけるCLI、Web、API間の分断、重複実装、認証・監査基盤の未統合、データモデル・API定義の欠如、保守・拡張コストの増大、UXの不整合といった課題の解決を目的としている。

「富岳NEXT」のUIは、API層を中心とする三層構造で設計される。GUIとCLIは共通のAPIを呼び出して機能し、出力整合性を保証する。これにより、GUIとCLIの操作は等価となり、新機能の追加や更新はAPIの拡張を通じて両者に即時反映される。API層では、ジョブ、課題、アカウント、利用統計などの共通データモデルを定義し、認証・認可・監査機能を統合的に取り扱うと共に、ユーザ作成ツールや外部システムから、ジョブ投入、データ転送、実行状況取得、モニタリング等を行うためのAPI群を標準提供する。バックエンドでは、運用データ基盤を用いてユーザ情報、課題情報、ジョブ情報、利用状況情報などを一元管理し、システム全体の整合性と再利用性を向上させる。

上記設計方針に基づき、富岳で顕在化した主要な課題に対しては以下の対応を図る。

- CLI と Web で重複・分断していた機能群の共通化: API モジュール化により共通化し、保守性を向上させる。
- 多言語・多方式で実装されていたコマンドおよびツール群の再構成: 統一的な開発規約と API ファースト原則に基づき再設計する。
- 性能向上: 性能要件を明確にし、非同期処理やキャッシュ機構を検討することで高速な API 応答を実現する。これにより、Web ポータルや CLI 操作時のレスポンス遅延を解消し、操作性の大幅な改善を図る。
- DevOps: 「富岳 NEXT」では、開発と運用の連携を強化する DevOps のアプローチを採用する。Git リポジトリ、CI/CD ツール、ドキュメントを一体的に管理する体制を構築し、自動テストと自動デプロイを導入することで、開発から運用までのライフサイクル全体を効率化する。これらには基本的に AI を利用することを前提とする。

上記より、「富岳NEXT」のユーザインターフェイスは、GUI、CLI、APIの3種類のインターフェイスを提供することを基本設計段階の前提とする。

一方、以下の事項については詳細設計段階に委ねる。

- GUI・Web 系の主要機能詳細
  ジョブ操作ポータル、可視化・監視ダッシュボード、実行環境管理機能、申請・問い合わせ機能、情報提供サイト（Docs / News / FAQ）など実装する機能とその詳細
- CLI 環境の主要機能詳細
  認証、ジョブ操作、課題管理参照、ファイルシステムアクセスなど実装する機能とその詳細
- API の主要機能詳細
  API 機能、レスポンスの形式などの仕様詳細
  ジョブ操作、課題管理、利用統計、監視メトリクスなどの取得対象

3.6.6.2 アプリケーション環境

「富岳NEXT」では、多様な利用分野のアプリケーションが最大限の性能と効率を発揮できるよう、既存アプリケーションとの互換性維持と、新たな利用への対応の両立を目指す。富岳向けに開発された多数の高性能アプリケー

ション資産を継承しつつ、ユーザが持ち込むあらゆるアプリケーションが円滑に実行可能な計算環境を実現する。

「富岳NEXT」のアプリケーション環境では、安定運用と利用者の自由度の両立が重要となる。特に ISV アプリケーションは依存環境やライセンス形態が複雑であり、動作保証にはベンダーサポートが不可欠であることから、システム側では FUJITSU-MONAKA-X で正式にサポートされたものを利用者が選択してコンテナ内で利用できるようにする。またライセンスは管理負荷と運用リスクを最小化するため持ち込みを基本とし、ワークロード管理下で資源として扱う場合に限りセンター管理とする。これにより、利用者固有のライセンス運用や外部連携の柔軟性を確保する。一方、ISV 以外のアプリケーションやワークフローは多様性が高く統一管理が困難であるため、導入・管理は利用者責任とし、システムの安定性と利用者自由度の双方を維持する。

上記理由から、基本設計として決定する事項は以下のとおりである。

- ISV アプリケーションについては、FUJITSU-MONAKA-X でサポートされるものを利用者が選択しコンテナ内で利用できるようにする。
- ISV ライセンスはシステムでは用意せず、ユーザが必要に応じて自ら持ち込むものとする。
- ISV ライセンスをワークロード管理下の資源として利用する場合のみ、ユーザのライセンスをセンターで管理するものとし、それ以外については利用者が管理するものとする。
- ISV 以外のアプリケーション（ワークフローなども含む）の導入や管理は、利用者の責任において実施する。

一方、以下の事項については詳細設計段階に委ねる。

- 利用者が選択できる具体的な ISV アプリケーション・バージョンの選定方法の検討
- 各 ISV ベンダーの FUJITSU-MONAKA-X への対応方針を考慮
- センター管理とするユーザのライセンスの持ち込み方法の検討
  ライセンスサーバの種類、ライセンスサーバへのアクセス管理、他利用者によるライセンス利用の抑止など
- 利用者によるライセンス管理の手段の検討
  テナント上でのライセンスサーバの立て方、外部のライセンスサーバ参照に必要なネットワークなど

3.6.6.3 パッケージ・モジュール管理

パッケージ・モジュール管理は、HPC 環境において多様なソフトウェア、ライブラリ、依存関係を整然と管理し、利用者が安定かつ再現性の高い実行環境を確保するために不可欠な仕組みである。HPC ではコンパイラ・MPI・数値計算ライブラリなど多様な組み合わせが存在し、従来の Linux 標準パッケージ管理では複雑性に対応できない場合が多い。バージョン共存、依存関係解決、環境再現性の確保は研究成果の信頼性に直結するため、体系的な管理基盤を備えたパッケージ管理とモジュール管理の両輪が必要となる。Spack のような柔軟なビルド管理と、Environment Modules による簡易な環境切替は、利用者の負担を軽減し、研究活動の効率を大きく向上させる。

「富岳NEXT」では、「富岳」で培われたSpack運用技術を基盤としつつ、HPC分野における最新のソフトウェアパッケージ管理ソフトウェアの技術動向も取り入れ、「富岳NEXT」の性能とソフトウェア管理効率を最大化する基盤技術を検討する。

3.6.6.3.1 特徴

SpackとEnvironment Modulesの特徴は以下の通り。

Spack の特徴

- 多様なバージョン・依存関係を自動解決してくれるため、複雑な環境構築が容易になる
- 高度なカスタマイズが可能で、最適化ビルドなど用途に応じた環境を構築できる

- 学習コストが高いため、初学者には使い始めのハードルが高い

Environment Modulesの特徴

- 導入・利用の習熟が容易で、HPC 初心者にもわかりやすい
- あらかじめ提供されたモジュールをロードするだけで、安全・確実に動く環境を得られる
- バージョンや依存関係の整合性はモジュールの設計側が担保するため、利用者の自由度は低め

3.6.6.3.2　本節で決定する事項と詳細設計に委ねる事項

上記より、利用者にはSpackとEnvironment Moduleの2つのソフトウェアパッケージ管理機能を提供することを基本設計段階における前提とする。

一方、以下の事項については詳細設計段階に委ねる。

- Spack（パブリック／プライベート）運用方式
- Environment Modules / Lmod の設計
- コンテナ環境との連携
- 運用・メンテナンス設計

3.6.6.4　Pre/Post 環境

Pre/Post は、利用者が計算ジョブを円滑に実行するために必要となる前処理および後処理を体系的に支援するユーザサービスである。前処理ではデータの整形、分割、入力準備、後処理では結果の抽出、集約、軽量解析、可視化準備など、計算そのもの以外に不可欠な作業を標準化し、利用者の作業負担を大幅に軽減する役割を担う。計算資源の有効活用とユーザ生産性の向上を実現するための重要なサービスである。

Pre/Postの実行環境として、計算ノードを用いる場合と、専用ノードを用意する場合の 2 パターンが考えられる。Pre/Post処理は、シミュレーション結果の価値を最大限に引き出すために不可欠な要素であり、特に「富岳NEXT」の時代において予想される解析の大規模化と複雑化へ対応できることが不可欠である。

- 大規模化への対応

  解析の大規模化によりデータ量は富岳時代と比較してさらに増加すると予想される。ペタバイト級、さらにはエクサバイト級のデータが生成されるシミュレーション環境では、Post 処理のワークフローでは I/O やネットワーク転送がボトルネックとして顕著になる。この問題を解決するためには、データが生成される場所、すなわち計算ノード上で Post 処理を実行できる必要がある。
- 解析対象の複雑化への対応

  Pre 処理においては解析の入力データや実行パラメータの組み合わせの決定において、Post 処理では多変量のシミュレーション結果から特徴の抽出において、推論による高度な支援が必要となる。例えば、多岐にわたる物理現象や化学反応を考慮した複雑なモデルでは、適切なパラメータ設定を見つけることが困難であり、AI/機械学習に基づく高度な推論能力による支援が不可欠となると考えられる。そのような推論を行う環境として計算ノードが最適である。

上記理由により、「富岳NEXT」のPre/Post環境は計算ノードを用いることを基本設計段階における前提とする。

一方、以下の事項については詳細設計段階に委ねる。

- 「富岳 NEXT」で実行される Pre/Post 処理の具体的ユースケースの整理
- 整理したユースケースを実現するために計算ノードに必要なシステムソフトウェアの調査
- 整理したユースケースを実現できる Pre/Post ソフトウェア（ISV, OSS 問わず）の調査
- 計算ノードの演算性能と並列 I/O を Pre/Post 処理にも利用したデータ処理効率化

- in-situ 処理や in-transit 処理といったソルバと Post 処理の Co-Processing（協調処理）により効率的に特徴抽出や画像生成を行うことによる、ストレージ負荷の削減

3.6.6.5 コンテナレジストリ

「富岳NEXT」では、コンテナを活用した多様なプロジェクトやジョブの効率的かつセキュアな運用を実現するため、独自のWebコンテナレジストリサービスを導入する。独自のコンテナレジストリを設ける主要な目的は、認証・署名されたコンテナの配布、データの機密性保持、外部通信によるトラフィックの低減、そして「富岳NEXT」全体での運用の一貫性とアカウントの共通化の実現にある。これにより、外部サービスに依存することなく、セキュアで高速なイメージ配信環境を提供し、サービスの信頼性を高める。

3.6.6.5.1 選択肢となる OSS

この新しいコンテナレジストリサービスを整備するにあたり、実現に用いるソフトウェアは必ずしもOSSとは限らないが、現状において利用可能なオープンソースソフトウェア（OSS）は、主に以下の通ものがある。

- Docker Registry (Distribution)
- Harbor
- Project Quay
- Zot

3.6.6.5.2 本節で決定する事項

本節において、基本設計として決定する事項は以下のとおりである。次の理由から Harbor と Zot を基本設計段階の候補として採用する。

- Harbor
  エンタープライズ用途で最も普及しており、セキュアなコンテナレジストリとして必要とされる機能を一通りそろえていることで、少ないコストでサービスを構築・運用できると考えられる。
- Zot
  軽量かつ OCIv2 互換であることから、今後コンテナレジストリソフトウェアとしての成長が期待でき、「富岳NEXT」が稼働する 2030 年以降においてモダンなコンテナレジストリサービスを構築できると考えられる。

3.6.6.5.3 詳細設計での検討事項

コンテナレジストリサービスを整備するにあたって、詳細設計で検討事項となるものを列挙する。

- 利用規模
  予想されるイメージ数、イメージサイズ、プッシュ/プル頻度を見積もる。
- 認証・セキュリティ
  認証方式、コンテナイメージへの署名や脆弱性スキャンなどサービスレベルでのセキュリティ事項を検討する。
- インフラ構成
- サーバ、ストレージ、ネットワーク、データベースなどインフラ部分について、サービス利用規模や可用性・セキュリティ要件にあわせた構成を検討する。
- 監視
- ストレージやネットワークなどインフラ共通の監視に加え、レジストリの状態、イメージ数、プッシュ・プル数などレジストリサービス特有の指標を監視する仕組みを検討する。
- 可用性

- サービスの可用性指標と目標値を検討する。

3.6.6.6 VPN 接続サービス

「富岳NEXT」では、研究者が学内外・国内外の多様な利用者環境から安全かつ効率的にシステムへアクセスできることが不可欠である。特に、データ駆動型研究・クラウド連携・外部施設との統合ワークフローが増加する中、外部環境と「富岳NEXT」を結ぶ専用ネットワーク経路（L2/L3 VPN 等）を前提とした接続方式が求められる。「富岳NEXT」の基本設計では、外部環境と「富岳NEXT」間のVPN接続サービスを提供することを基本設計段階での前提とし、詳細設計以降でVPN種別（L2/L3、暗号化方式）、帯域、可用性設計などの具体的な検証を行う。

3.6.6.7 外部サービス利用

AI・データ駆動型研究および複合ワークフローの高度化により、外部クラウドサービス、外部 APIとの連携は極めて重要な利用形態となっている。「富岳NEXT」では、こうした外部サービスとの安全かつ柔軟な連携を可能にすることで、研究者の生産性向上・分析効率化・外部データ基盤との統合など、多様な研究要求に応える必要がある。「富岳NEXT」の基本設計では、外部クラウド・外部 API・外部計算サービスの利用を認めることを基本設計段階での前提とする。

詳細設計以降の検討項目について、以下に述べる。

- 外部クラウド・API・計算サービスとの具体的な接続方式とネットワーク構成
- 接続先制御（Outbound Allowlist 等）の詳細
- 外部サービス利用時のセキュリティ要件・監査ログ管理
- データ転送方式・帯域制御・QoS

3.6.6.8 サービスカタログ

サービスカタログは、「富岳NEXT」が提供する各種サービスを体系的にまとめ、利用者が利用可能な機能、性能目標、制限、セキュリティ方針、ネットワーク要件、API連携、コスト配分、ライフサイクル管理などを明確に示す文書である。これにより、利用者は提供範囲・利用条件・品質レベルを正しく理解でき、運用側は全利用者に対して一貫したサービスを提供可能となる。また、サービスの透明性・公平性を確保し、問い合わせ削減やトラブル予防にも寄与する。複雑化する「富岳NEXT」の多様なサービスを整理し、利用者体験の向上と運用効率化を同時に実現する基盤として不可欠である。上記理由から、「富岳NEXT」では、サービスカタログを整備することを基本設計段階における前提とする。

一方、以下の事項については詳細設計段階に委ねる。

- サービス概要（例：名称、目的、対象ユーザ、提供価値）
- サービス内容（例：提供機能、サービス構成要素、利用シナリオ、依存サービス）
- 利用条件・サービスレベル（例：提供時間、性能・可用性・容量、制約事項）
- 利用手続き・サポート（例：申請方法、承認フロー、問合せ窓口、バックアップポリシー）

3.6.6.9 ユーザ向けドキュメント

ユーザ向けドキュメントは、「富岳NEXT」の利用方法を段階的かつ体系的に習得するための教材群であり、クイックスタート、ユーザガイド、データ転送手順、プログラミング環境、GPU移行、コンテナ操作、可視化、API利用など多様な内容を含む。高度化したHPC＋AI基盤を円滑に使い始めるための実践的な支援材料であり、利用者が自力で問題を解決しやすくなる点に意義がある。これによりサポート負荷の軽減、研究作業の迅速化、利用者体

験の向上が期待でき、「富岳NEXT」の有効活用に必須の要素である。

3.6.6.9.1　ドキュメントの提供形態

以下に、ドキュメントの提供手段として想定される媒体を示す。

- PDF 提供

  概要：ひとつの完成された冊子（または複数 PDF）として配布する形式。

  利点：静的な安定版として管理可能。オフライン閲覧や印刷物として整った体裁。

  欠点：更新ごとに改訂版の作成が必要。検索性・情報鮮度に劣る。閲覧性がデバイス依存。
- Web(HTML)提供
- 概要：Web サイト上で階層化した章・ページとして提供する形式。

  利点：更新が即時・部分的に可能。検索性・導線に優れる。Web ポータルとの連携が容易。アクセス解析が可能。

  欠点：Web システムの保守が必要。オンライン前提。レイアウトに制約。

以下に、ドキュメントの体系化の構成として想定されるものを示す。

- 分冊構成

  概要：全ての情報を 一つの大きな文書（ファイル、もしくは Web ページ）にまとめる方式。

  利点：部分的更新が容易。利用者のニーズに応じた入口を提供。文書構造を柔軟に拡張可能。

  欠点：情報が散逸する。冗長化や重複記述が発生する。全体像の把握に向かない。
- 集約構成

  概要：情報を 複数の冊子（ページ）に分解し、用途別・対象別に分けて提供する方式。

  利点：全体像が把握しやすい。記載内容の整合性が取りやすい。参照関係がシンプル。

  欠点：全体更新が必要。ユーザが必要箇所を探しにくい。構造拡張がしにくい。

3.6.6.9.2　本節で決定する事項

本節において、基本設計として決定する事項は以下のとおりである。

- 「富岳 NEXT」では、ユーザドキュメントを作成し提供する。
- 可読性を考慮し、ドキュメント媒体は Web、細分化せず集約構成で提供する。

3.6.6.9.3　詳細設計以降の検討項目

詳細設計以降では、以下の項目について検討する。

- ドキュメントの提供時期（プログラミングガイドの早期公開の可否など）
- ドキュメントの体系（クイックスタート、プログラミングガイドなど）
- トレーニングでの活用方針（チュートリアル、システム導入前教育など）

3.6.6.10多言語対応

本節では、「富岳NEXT」における多言語対応の方針を基本設計として定義する。「富岳NEXT」のサービスインターフェース（Webサイト、システム、ドキュメント類など）、およびテクニカルサポートは、日本語と英語を公式にサポート言語とする。ただし、どちらか一方を選択する場合は英語とすることを基本設計段階の前提とする。この方針は、日本が開発・運用を主導しつつも、世界中の研究者による利用を想定しているためである。これにより、主要な利用者層が言語の障壁を感じることなく、「富岳NEXT」の機能や情報を円滑に活用できる環境を提供する。

3.6.7 運用支援システム

「富岳NEXT」における運用支援システムとは、利用者のHPCシステム利用における各種手続き（申請、情報参照、問い合わせなど）から、運用管理者によるシステム運用管理（ユーザ・課題・利用資源管理、課金、利用統計など）までを一元的かつ効率的に支援するシステム群である。

「富岳NEXT」では、利用者利便性の向上およびシステム運用効率化を目的として、運用支援システムを導入する。

3.6.7.1 要求事項を踏まえた設計方針

本節では、「富岳」からの課題および2030年を見据えた要求事項に対応するための、本基本設計段階における設計方針を記述する。本方針に基づき設計を実施することで、将来的な制度変更に柔軟に対応可能なシステムの設計を目指す。

- 運用の自動化と省力化
  24 時間 365 日の安定稼働と運用負荷軽減のため、利用申請からアカウント等の管理、簡易な問い合わせ対応に至るまで、可能な限り自動化する。またこの自動化には AI を用いる。
- ユーザエクスペリエンスの向上
- 煩雑な複数システムへのアクセスを解消し、システム利用に関するあらゆる操作（申請、情報参照、問い合わせ など）において、直感的で利用しやすい UI/UX を設計する。
- モジュール性と疎結合性
  運用制度や利用ポリシーの変更に対し、ツールの改修コストを最小限に抑え、運用開始後も迅速かつ柔軟に運用要員のみで対応可能なアーキテクチャと機能設計を検討する。実際の開発・プログラミング自体は AI が行うことを想定した体系を整備する。
- 高い信頼性、可用性、スケーラビリティ
  「富岳 NEXT」がミッションクリティカルな社会インフラとなることを鑑み、運用支援システム自体も高い信頼性と可用性を持つよう冗長化を前提とする。また、将来的な利用者数・利用種類の増加に対応できる高いスケーラビリティを確保する。
- HPCI とのシームレスな連携
  現行「富岳」での HPCI 連携をベースに、HPCI 統一アカウントとの連携を円滑に行い、HPCI システムの一部として機能することを要求する。

3.6.7.2 開発項目

本節では、詳細設計段階にて検討および決定する事項を記述する。

3.6.7.2.1 機能要件

利用者利便性の向上とシステム運用の効率化を目的として、以下の機能要件を定義する。

ユーザ管理機能では、アカウントの発行・管理・認証、権限付与を担い、アカウント登録や利用期間延長等を実現する。課題管理機能では、資源割り当てとアクセス制御を効率化する。利用資源管理機能では、HPC資源の公平な配分と利用状況の可視化を行う。課金システムは、課金データ自動計算・請求書生成を行い、利用料金の透明性と管理を向上させる。利用統計機能は、運用データ基盤からのデータ収集と分析を通じて、システム利用状況を可視化し、運用改善に貢献する。チケットシステムは、問い合わせ受付、FAQ、チャットボットを統合し、利用者サポートの迅速化を図る。HPCI連携機能は、HPCIアカウントや利用計画／実績データの同期・連携により、既存のHPCIエコシステムとのシームレスな統合を実現する。

3.6.7.2.1.1　ユーザ管理機能

アカウントの発行・管理、認証、ロール（権限）の付与・管理を行う。主な機能は、以下の通り。

- アカウント登録
  利用者からの申請に基づき、新規ユーザのアカウントをシステムに登録する。
- アカウント更新
  登録済みのアカウント情報（所属、連絡先など）を変更する。
- アカウント利用停止/解除
  アカウントの一時的な利用停止や設定解除を行う。
- アカウント削除
  不要となったアカウント情報をシステムから完全に削除する
- アカウント利用期間延長
  利用期間が終了するアカウントの利用可能期間を延長する。

3.6.7.2.1.2　課題管理機能

課題単位でのグループ作成、メンバーの追加・削除、代表者・管理者の設定。課題毎の資源割り当てやアクセス制御を行う。主な機能は、以下の通り。

- 課題登録
  新規の研究課題をシステムに登録する。
- 課題変更
  登録済みの課題情報（所属メンバ、割り当て資源量など）を変更する。
- 課題継続
  課題の期間を延長し、関連情報を更新する。
- 課題削除
  終了した課題情報をシステムから削除する。

3.6.7.2.1.3　利用資源管理機能

CPU/GPU時間、メモリ、ストレージなどのHPC資源の使用状況を収集し、公平な配分を保証する。収集データは、課金システムと連携する。管理対象の利用資源量は、詳細設計段階にて検討する。主な機能は、以下の通り。

- リソース割り当て
  課題に対し、資源量を配分する。
- リソース申請/承認ワークフロー
  課題代表者がリソースの拡張を申請し、管理者による審査・承認を管理する一連のプロセス。
- リソース利用状況表示
  現在のリソース消費状況や残量を可視化して提示する。
- リソース利用上限設定
  課題や利用者に割り当てるリソース量の上限値を設定し、消費を管理する。

3.6.7.2.1.4　課金システム

利用資源管理で収集されたデータに基づき、事前に設定された課金レートで利用料金を計算し、課金請求情

報を作成する。詳細設計段階にて、課金対象の利用資源と仕組みを具体化する。主な機能は、以下の通り。

- 改札制御
  利用可能予算を管理し、予算超過時は利用者に通知する。
- 課金計算
  利用された資源量、ポイントなどに基づいた課金計算を行う。
- 課金明細/請求書生成
  ユーザ/グループごとの利用明細、請求書を Web 表示する。
- 課金 RATE 設定
  専有利用などの様々な運用施策実現のための連携機能。
- 電力連携
  「富岳ポイント」のような制度実現のための連携機能。

3.6.7.2.1.5　利用統計機能

運用データ基盤と連携し、システム稼働率やジョブ統計などのデータを分析し、ダッシュボードで可視化する。

- 利用統計データ分析
  システム全体の利用率、ジョブ実行数、ユーザ別/グループ別のリソース利用量、ジョブ性能などのデータを分析。
- 統計データ分析/可視化
  収集されたデータを分析し、ダッシュボードでグラフ表示、レポート生成。

3.6.7.2.1.6　チケットシステム

問い合わせや障害報告受付からFAQ、チャットボットによる自動応答、チケット管理まで、利用者サポートを一元化する。

- 問い合わせ/障害報告受付
  ユーザからの問い合わせ、障害報告、要望を一元的に Web 上で受け付け
- FAQ/ナレッジベース
  よくある質問 (FAQ) とその回答を体系的に蓄積・公開
- チャットボットによる自動応答
  自然言語処理を活用し、簡易な問い合わせや FAQ に関連する質問への自動応答
- チケット管理
  チケットのステータス管理、履歴管理

3.6.7.2.1.7　HPCI 連携機能

HPCI各種制度の下で円滑な運用を実現するため、ユーザ情報や利用状況をシームレスに連携させることで、広範なHPCIエコシステムへの統合を図る。

- HPCI アカウント連携
  ユーザ情報連携

3.6.7.2.2　非機能要件

信頼性と持続可能性を確保するために、以下の非機能要件を定義する。

性能要件として、ユーザインターフェイスの応答速度向上、ピーク時の高い処理スループット、統計データを日次で

高速処理する能力が求められる。可用性要件では、計画メンテナンスを除く高い稼働率を保証し、Webサーバ、DBサーバ、アプリケーションサーバー等の主要コンポーネントの多重化による単一障害点排除を要求する。拡張性要件には、利用者数やデータ量増加へのスケーラビリティへの柔軟な対応力が含まれる。保守運用要件では、操作履歴・イベントのログ管理、定期的なバックアップと迅速なリカバリ、最小限のサービス停止でのパッチ適用・アップデートを要求する。セキュリティ要件としては、役割ベースのアクセス制御（RBAC）、定期的な脆弱性診断によって、システムの安全性と信頼性を高める。

3.6.7.2.2.1　性能要件

ユーザ操作への迅速な応答速度、ピーク時の高い処理スループット、大規模な統計データを日次で高速処理し、分析結果を速やかに表示する能力を要求する。

- 応答速度
  ユーザインターフェイスからの一般的な操作（ログイン、情報参照、簡易申請など）に対する応答時間
- 処理スループット
  ピーク時における申請処理件数、問い合わせ受付数
- データ処理量
  大規模な統計データを日次で処理し、分析結果を速やかに表示できること。

3.6.7.2.2.2　可用性要件

計画メンテナンスを除く高稼働率、主要コンポーネントの多重化による単一障害点排除を要求する。

- 稼働率
  年間のシステム稼働率（計画メンテナンスを除く）。
- 冗長化
  主要なコンポーネント（Web サーバ、DB サーバ、アプリケーションサーバーなど）は多重化し、単一障害点を排除する。

3.6.7.2.2.3　拡張性要件

リソース追加や分散配置でトラフィック増に対応するスケーラビリティと、新技術・機能追加に既存システムへの影響を抑えつつ対応する柔軟性を要求する。

- スケーラビリティ
  利用者数、同時接続数、データ量の増加に対し、容易にリソースの追加や分散配置により対応できるアーキテクチャであること。
- 柔軟性
  新しい HPC 技術や利用形態（例: 量子コンピュータ連携、新しい AI フレームワーク）の導入、機能追加に対して、既存システムへの影響を最小限に抑えつつ対応できること。

3.6.7.2.2.4　保守運用要件

監査や障害解析に利用可能な操作履歴・エラーのログ管理と、データ破損時に迅速な復元を可能にする定期的なバックアップ/リカバリ、サービス停止時間を最小限に抑えたパッチ適用・アップデートを要求する。

- ログ管理
  システムのあらゆる操作履歴、エラー、イベントを記録し、監査や障害解析に利用できること。ログは適切な期間保存されること。

- バックアップ/リカバリ
  全てのシステムデータは定期的にバックアップされ、データ破損時には速やかに復元できること。
- パッチ適用/アップデート
  セキュリティパッチやソフトウェアアップデートを、サービス停止時間を最小限に抑えつつ適用できる仕組みを有すること。

3.6.7.2.2.5　セキュリティ要件

役割ベースのアクセス制御による厳格なアクセス制限、定期的な脆弱性診断を通じてシステムの安全性を要求する。

- アクセス制御
  役割ベースのアクセス制御 (RBAC) を実装し、未認証・未認可のアクセスを厳密に制限する。
- 脆弱性対策
  定期的な脆弱性診断を実施する。

3.6.7.3　本章で決定する事項と詳細設計に委ねる事項

本節において、基本設計として決定する事項は以下のとおりである。

- 運用支援システムを導入すること
- 運用支援システムへの要求事項への対応として、「運用の自動化・省力化」「ユーザエクスペリエンスの向上」「モジュール性」「高信頼性・可用性・スケーラビリティ」「HPCI 連携」を核とした設計方針とすること

一方、以下の事項については詳細設計段階に委ねる。

- 開発項目における機能要件、非機能要件
- システム概要、システム仕様、ソフトウェア選定、実装方法
- 運用ポリシー、運用手順

3.6.8 運用基盤

「富岳NEXT」では、現行「富岳」における運用実績を踏まえ、4,000台を超えるハードウェア、およびハードウェア上で動作する各種ソフトウェアを安定して稼働させる運用基盤の設計を目指す。基本設計では運用基盤として検討すべきソフトウェアをプロビジョニングソフトウェア・構成管理ソフトウェア・監視ソフトウェア・運用データ基盤ソフトウェアの4つに分類し、「富岳NEXT」における検討項目、およびソフトウェアを提示する。

システム運用を実現する以下のソフトウェアの検討、および、これらソフトウェアを用いて効率的・安定的にシステム運用する為の検討項目について記載する。

- プロビジョニングソフトウェア
  ハードウェア（サーバ）に対して OS のインストール・アンインストールを実施
- 構成管理ソフトウェア
  ハードウェアの設定変更、およびハードウェア（サーバ）上で動作する OS／ミドルウェア／アプリケーションのインストール・アンインストール・設定変更を実施
- 監視ソフトウェア
  ハードウェア、およびハードウェア（サーバ）上で動作するソフトウェアの情報を収集し、異常検知を実施
- 運用データ基盤ソフトウェア
  ハードウェア、およびハードウェア（サーバ）上で動作するソフトウェアの情報の収集・保存・管理を実施

3.6.8.1 プロビジョニングソフトウェア

本節では、プロビジョニングソフトウェアについて検討する。プロビジョニングとは物理サーバにOSをインストールして利用可能な状態にするための作業であり、この作業を自動化・効率化するソフトウェアをプロビジョニングソフトウェアと呼ぶ。

「富岳NEXT」では大規模ノードを迅速に構築する必要があることから、サーバへのプロビジョニング作業の自動化・効率化が求められる。現状の「富岳」運用においてプロビジョニング作業はFujitsu Software Technical Computing Suiteによって行われているが、冒頭に挙げられた「富岳」システムでの課題に対応する為、それに代わるソフトウェアの選定が必要となる。

基本設計においては、OSプロビジョニング時における各種基盤コンポーネント(DHCP, TFTP等)との連携、および大規模環境への対応実績の観点から、プロビジョニングソフトウェアとして、ForemanとOpenCHAMIの2つを採択し、これらのソフトウェアの詳細な検討は詳細設計にて行う。

以下にソフトウェアの採択理由と、詳細設計の検討項目を示す。

- ソフトウェアの採択理由
  プロビジョニングソフトウェアの比較結果を踏まえると Confluent と Warewulf は、OS プロビジョニング時に DHCP・DNS といった基盤コンポーネントとの連携ができない為、プロビジョニング前後で基盤コンポーネントの設定変更を別途行う必要があり、大規模環境でプロビジョニングを並行して行う際の操作が煩雑になる他、設定の不整合が生じやすい。一方で、Foreman と OpenCHAMI は機能項目での懸念点は無いが、複数サブネットに分散した計算ノードへのプロビジョニングをサブネット内に閉じて行える点からシステム構成の柔軟性は Foreman の方が高い。又、スケーラビリティ観点では大規模環境での動作実績が Foreman の方が優れている他、Foreman にはプラグイン形式での機能拡張の仕組みがある等、機能拡張性も考慮されている。上記の様な差異が Foreman・OpenCHAMI 間にあるものの機能項目での問題点は無いため、基本設計では Foreman・OpenCHAMI をプロビジョニングソフトウェアとして採択することとする。

- 詳細設計の検討項目
  詳細設計では Foreman・OpenCHAMI に関して、機能差異の実運用への影響含め、調査・検討する。以下に詳細設計の検討項目を示す。
  - 採用するプロビジョニングソフトウェアの決定
  - 「富岳 NEXT」システムの運用要件の明確化と、要件に照らし合わせた際の選定したソフトウェアの機能面・非機能面での充足性の検証

3.6.8.2 構成管理ソフトウェア

本節では、構成管理ソフトウェアについて検討する。構成管理とはハードウェア・ソフトウェアが提供するインターフェイスや機能を用いて、システム運用に必要なハードウェア・ソフトウェアに対する各種操作(構成管理)を実施する作業である。これら作業を統一的に管理し自動化・効率化するソフトウェアを構成管理ソフトウェアと呼ぶ。又、構成管理ソフトウェアの中でも、ハードウェアへの操作に閉じるものをハードウェア構成管理、ソフトウェアへの操作に閉じるものをソフトウェア構成管理と呼ぶ。

「富岳NEXT」では従来のHPC処理の他にAI処理等、利用用途の拡大が想定されていること、また、これらのソフトウェアの管理は従来のパッケージやコンテナ（旧Singularity）によるものだけではなく、Kubernetes等の新たな基盤も想定されていることから、システム構成がますます複雑になる。この複雑化するシステムにおいて、ハードウェア・ソフトウェアへの種々の操作とシステムの設定/構成変更を効率化するためには構成管理ソフトウェアの選定が必要となる。

基本設計においては、構成管理対象のハードウェア・ソフトウェアの種類、構成管理の定義の容易性、および、実行履歴の管理を考慮し、構成管理ソフトウェアとしてAnsible・AWXを採択することとする。なお、AWXにおいては2024年よりコミュニティで大規模な改修が行われているという懸念点があるため、継続してコミュニティの活動状況を注視する。これら採択したソフトウェアの詳細な検討は詳細設計にて行う。

以下にソフトウェアの採択理由と、詳細設計の検討項目を記載する。

- ソフトウェアの採択理由
  構成管理ソフトウェアの比較結果を踏まえると Chef と Puppet は、システム構成定義の記述に独自の DSL を使用しており習熟が必要な点に懸念がある。一方で、Ansible と Salt は YAML を用いている為、システム構成定義を記述する敷居が低い他、機能項目の比較においても特に懸念点は無い。Ansible と Salt 間では GUI の開発主体や実行履歴の管理に差異があり、コミュニティが公式に開発をしている AWX にて GUI による構成管理の実施と実行履歴を管理できる点が Ansible の特徴である。又、想定される構成管理で行う操作のサポートは Ansible が特出している。その為、構成管理ソフトウェアとして Ansible を採択することとする。
- 詳細設計の検討項目
  詳細設計では Ansible を主として、詳細な構成管理項目・操作への対応可否について調査・検討する。以下に、詳細設計の検討項目を示す。
  - 「富岳 NEXT」システムを構成するハードウェア・ソフトウェアの決定に伴う、システム運用で必要な構成管理項目とその対象の整理および、構成管理実現手法の決定
  - 「富岳 NEXT」システムの運用要件の明確化と、要件に照らし合わせた際の選定したソフトウェアの機能面・非機能面での充足性の検証

3.6.8.3 監視ソフトウェア

本節では、監視ソフトウェアについて検討する。監視ソフトウェアとは、システムおよびシステム上で動作するソフトウ

ェアの稼働情報(システムデータ)を基にシステムの状態を監視し、システム内の異常検知・システム稼働状況の可視化をするソフトウェアである。

「富岳NEXT」では利用用途の拡大に伴い、それらの処理を実現する為のハードウェア・ソフトウェアの種類が多様化し、システム構成がますます複雑になる。その様な環境においてシステム障害発生時の早期対応やシステムの安定稼働を実現するためには、「ハードウェアとソフトウェアの監視・異常の検知および通知・稼働状況の把握」を実現する監視ソフトウェアの選定が必要となる。

監視ソフトウェアは、システムデータを収集・保存する「システムデータ収集保存機能」と、システム監視とシステム可観測性を実現する「監視機能」から構成される。

- システムデータ収集保存機能
  システム内のハードウェアおよびソフトウェアの情報(システムデータ)を収集し保存・管理する機能。情報の取得・生成を行う「システムデータ生成・取得部」、および収集したデータを保存・管理する「システムデータ保存・管理部」で構成される。
- システム監視
  収集されたシステムデータを基に、システム内の正常・異常イベントの発生有無を検知し、検知した際は電子メール等の手段を通して外部に通知する
- システム可観測性
  収集されたシステムデータを用いて「システムの稼働状況の把握」と「システムを構成する各コンポーネントの挙動の把握」を容易に行えるようにする

基本設計においては、「システムデータ生成・取得部を実現するソフトウェア」・「システムデータ保存・管理部を実現するソフトウェア」・「システム監視を実現するソフトウェア」・「システム可観測性を実現するソフトウェア」を検討し、以下に記載のソフトウェアを調査することとする。これらソフトウェアの詳細な検討は詳細設計にて行う。

監視ソフトウェアで実現する機能	**機能を実現するソフトウェア**	
システムデータ収集保存機能	システムデータ生成・取得部	OpenTelemetry Fluent Bit Prometheus Exporter Beats Elastic Agent
	システムデータ保存・管理部	Prometheus Elasticsearch OpenSearch
監視機能	システム監視	OpenSearch Kibana Alertmanager
	システム可観測性	OpenSearch Kibana Grafana

3.6.8.4 運用データ基盤ソフトウェア

運用データ基盤とは「富岳NEXT」に構築されたシステムおよびシステム上で動作するソフトウェアの稼働情報(シ

ステムデータ)やジョブ統計情報などの利用者データを蓄積・分析することで長期的な改善を実現する基盤である。
運用データ基盤における、具体的な管理対象やソフトウェアの検討は詳細設計で行う。

3.6.9 セキュリティ

3.6.9.1 セキュリティポリシー

「富岳NEXT」は、国内外の多様な研究者が利用する大規模共用計算基盤であり、安全性・可用性・信頼性の確保は最重要の設計要件である。計算ノード・ストレージ・ネットワーク・コンテナ基盤が密接に連携する次世代 HPC では、単一の防御策ではなく、ネットワーク分離、テナント隔離、アクセス制御、監査・ログ、運用監査などを組み合わせた多層的なセキュリティが不可欠である。本節では、「富岳NEXT」の基盤となる「分離構造」「ネットワーク設計」「テナントモデル」「監査体制」「ログ運用」について整理し、安全性と柔軟性を両立するための基本的考え方を示す。

3.6.9.1.1 検討内容

セキュリティポリシーは、「富岳NEXT」全体の構造に関わるため、基盤技術・外部事例・国際標準・運用要件を踏まえて多角的に検討する必要がある。

- ネットワーク構成分離
  「富岳 NEXT」では、多数の利用者が同時に利用することを考慮し、ネットワークの分離を基盤的な要件として整理する必要がある。管理ネットワークと利用者のデータ通信を混在させることはセキュリティリスクを増大させるため、物理・論理の双方で厳格に分離する設計が有効であると確認された。また、外部との通信経路を最小化し、必要な場合のみ適切な制御と監視を行うことで、外部脅威に対する防御力を高めることができる。
- テナント分離方式
  プロジェクトごとに異なる利用者・異なるワークロードが実行されるため、それぞれが互いに干渉しない実行環境を提供する必要がある。コンテナ基盤を活用したテナント分離が、計算リソースとネットワークリソースを同時に隔離でき、柔軟性と安全性を両立できる方式として有効である。また、VPN による利用者環境も含めたテナントを閉域化することで、セキュリティを確保した実行環境を提供することが可能となる。
- ストレージのアクセス分離方式の調査
  ストレージについては、テナント隔離の構造と整合させる必要がある。利用者ホーム／プロジェクトディレクトリをテナント環境に安全にマウントすることで、参照可能パスを制御し他プロジェクト領域を不可視化することでセキュリティを確保することが可能となる。
- 監査・ログ設計
  安全な運用とインシデント対応には、全レイヤの行動を可視化し、必要なログを統合的に収集・保存する仕組みが不可欠である。特に、認証・認可、ジョブ実行、ネットワーク通信、システム操作などのログは、状況把握や調査分析に必要となる。また、長期保存と改ざん防止を考慮した運用設計も重要である。
- セキュリティホワイトペーパーの検討
  利用者・外部組織に対し、「富岳 NEXT」の安全性を透明性高く説明するため、セキュリティアーキテクチャを文書化したホワイトペーパーが必要である。これは利用者の安心感を担保するだけでなく、外部連携における信頼性の証明にも役立つ。

3.6.9.1.2 本節で決定する事項と詳細設計に委ねる事項

本節において、基本設計として決定する事項は以下のとおりである。

- コンテナによりプロジェクトごとに閉じたネットワーク空間を持つテナントを構築する。
- ストレージは利用者ディレクトリをコンテナにマウントする方式とする。

- 運用開始時にセキュリティ監査を実施する。
- 必要な監査ログを取得し保存する方針とする。
- セキュリティホワイトペーパーを作成する。

一方、以下の事項については詳細設計段階に委ねる。

- テナント実現方式（ネットワークポリシー・CNI 設定・サービスメッシュ等）
- ストレージマウント方式の詳細（アクセス権、暗号化併用、パス制御）
- セキュリティ監査の頻度・実施項目・実施体制
- 保存するログの種類・保存期間・アクセス権の詳細設計
- ホワイトペーパーの項目・更新方針

3.6.9.2 認証・認可

「富岳NEXT」では、Web ポータル・SSH・API・ワークフロー基盤など多様な利用経路を統合的に扱う必要があるため、認証方式と認可体系の一貫性が重要である。また、アカウント管理・アクセス制御・多要素認証・外部連携といった幅広い論点を含むため、統合認証（SSO）、認可モデル（RBAC 等）、最新認証方式について整理する。これらは、利用者体験向上に加えて、セキュリティと運用効率の両立に不可欠である。

「富岳NEXT」は従来富岳の複数認証基盤の課題を解消し、単一・最新の認証基盤へ統合することで、権限不整合の防止、管理効率化、利用者負担の軽減を図る必要がある。

上記理由より、SSHとWEBなどを含め最新の認証方式を用いた統合認証を行い、SSO（シングル サインオン）環境を構築することを基本設計段階における前提とし、SSOに用いる認証・認可技術やSSHとWEBの統合認証方式などについては、詳細設計以降での検討事項とする。

3.6.9.3 セキュリティインシデント対応

「富岳NEXT」は、多数の利用者と多様なワークロードが同居する大規模計算基盤であり、システム障害・攻撃・誤操作などのインシデントは、サービス停止や研究データの損失、外部への影響拡大につながる可能性を持つ。特に、計算資源・ストレージ・ネットワーク・外部接続が密接に連動する環境では、インシデント発生時の判断や対応の遅れが、被害範囲を一気に拡大させる恐れがある。そのため、インシデントの種類、初動対応、組織的な役割、証跡管理、復旧および改善といった観点から、体系的なインシデント対応の枠組みを整理し、「富岳NEXT」に求められる対応体制を明確化する必要がある。

上記理由より、インシデント発生時に迅速に復旧・対応可能となるよう、インシデント対応手順を策定することを基本設計段階における前提とし、対応手順に含める項目、役割分担・指揮系統・外部報告フローなど体制、訓練体系・教育プログラム・改善サイクルの具体化などについては、詳細設計以降での検討事項とする。

3.6.9.4 暗号化

「富岳NEXT」では、多様な研究データ・個人情報・機密性の高い成果物が扱われ、さらに外部接続やクラウド連携など利用形態が高度化するため、通信および保存データに対する暗号化の実施は安全性確保の上で不可欠である。また、暗号化は単一要素ではなく、通信経路・保存データ・鍵管理の三点を統合的に扱う必要があり、それぞれの方式の選定や適用範囲の判断は、セキュリティレベル・性能影響・運用負荷を踏まえて整理する必要がある。

「富岳NEXT」では、攻撃者による盗聴・改ざん・不正アクセス、内部からの誤操作や不適切な取扱いなど、多様なリスクを低減するために、暗号化を全体アーキテクチャに組み込むことが求められる。特に、計算ノード・ネットワーク・ストレージが複雑に連携する大規模システムでは、暗号化を適切に適用する箇所を明確化し、鍵管理方式

を含めた一貫した運用モデルを確立することが重要である。

上記理由より、「富岳NEXT」では必要な箇所への暗号化を実施することを基本設計段階の前提とし、どこを暗号化し、どの方式を採用し、どのように鍵管理を行うかについては、性能要件・運用制約・利用形態を踏まえ、詳細設計以降で検討する事項とする。

3.6.10運用プロセス

「富岳NEXT」の運用は、計算資源の高効率利用、サービス品質の維持、障害の迅速な復旧、セキュリティ確保など、多岐にわたる要求を安定的に満たす必要がある。特に、AI・データ駆動型ワークロードの増加、外部基盤との連携、高可用性・可観測性の強化といった要件が高まる中、運用プロセスはシステムの信頼性と持続性を担保する基礎となる。このことから、「富岳NEXT」においてはITIL や ITSMOP 等の標準に沿った体系的な運用プロセスを策定することを基本設計における決定事項とする。これにより、運用の属人化を防ぎ、継続的改善が可能な運用体制の構築を支援する。実際にはAIを活用した運用自動化を前提として体系化する。

詳細設計にて検討する項目は以下の通りである。

- 運用体制
  運用体制の責任範囲・権限・エスカレーションをどのように具体化し、日常運用に落とし込むかを検討する。運用組織の役割分担、権限、担当領域を明確化し、エスカレーションルールを定義する。
- 保守・メンテナンス管理
  計画停止・緊急停止の基準、作業フロー、影響評価、通知方法をどのように標準化するかを検討する。定期・臨時メンテナンスの手順、通知方法、ジョブ調整方式を策定し、影響最小化を図る。
- インシデント管理
  インシデント検知手段、対応プロセス、優先度分類、連絡フロー、復旧判断基準をどのように定義するかを検討する。障害検知から復旧、再発防止までの一連の流れを標準化し、影響範囲特定と対応速度を向上させる。
- 問題管理
  根本原因分析手法、問題登録基準、永続的対策の実施プロセスをどのように組み込むかを検討する。インシデントの根本原因分析を体系的に行い、構成改善・対策案を継続的に反映する仕組みを整備する。
- 変更管理
  変更申請、影響分析、審査・承認、実施、検証、ロールバックの各工程をどのように標準化するかを検討する。OS 更新、設定変更、構成更新などの変更を計画的に実行し、事前審査・影響分析・承認プロセスを整備する。
- サービスカタログ管理
  サービス項目、提供条件、制限事項、SLO をどのように定義し、利用者に提示するかを検討する。利用者向けサービス内容、SLO、制約条件を整理し、サービス提供範囲を明確化する。
- SOP（標準運用手順）整備
  作業手順書の粒度、更新手順、CI/CD 連携による手順自動更新の方式を検討する。運用手順を標準化し、作業再現性と運用品質を確保する。CI/CD と連動した更新管理も検討する。
- 検証手順・リリース管理
  検証環境の構成、テスト項目、リリース判定基準をどのように体系化するかを検討する。本番適用前の検証環境整備、構成差分管理など、リリース品質を維持するための基盤を整備する。
- 文書管理
  作成対象文書、更新手順、版管理方式、保管場所、アクセス権限をどのように設計するかを検討する。構成図、手順書、運用台帳、変更履歴などの文書体系を整備し、変更に追従した更新を行う仕組みを構築する。
- SLA/SLO 管理
  監視指標、収集方法、レポート形式、閾値設定、改善プロセスをどのように確立するかを検討する。サービス品質指標（可用性、応答時間、性能等）を定義し、SLO 監視・レポート作成方式を策定する。

- データ保護・BCP/DR
  バックアップ構成、保管期間、復旧手順、DR サイト運用方式をどのように定義するかを検討する。バックアップ方針、データ保持ポリシー、DR 手順などを整理し、災害時や重大障害時の業務継続性を確保する。
- トレーニング・ナレッジ管理
  教育体系の構造、トレーニング実施計画、ナレッジ蓄積と検索性向上の方法を検討する。運用者向け教育体系、ナレッジベース整備、運用知識の継続的蓄積方法を策定する。
- 監査・遵守事項
  監査ログの範囲、保管期間、アクセス統制、監査対応プロセスをどのように設計するかを検討する。内部・外部監査に対応可能なログ管理、証跡保持、変更履歴の管理基盤を整備する。
- ライフサイクル管理
  導入・更新・廃止の各段階で必要となる記録・承認・作業プロセスをどのように体系化するかを検討する。システム導入から廃止までの全期間にわたる構成・サービス管理プロセスを整理する。
- 運用評価と改善（Continual Improvement）
  評価指標、レビューサイクル、改善提案の取り込み方法、プロセス改善の継続管理方式を検討する。KPI/SLO の定期評価とプロセス改善ループを運用に組み込み、継続的最適化を図る。

3.6.10.1 システム導入

システム導入計画は、「富岳NEXT」の導入を円滑かつ安全に進めるために不可欠である。ハードウェア搬入から構築・試験、ユーザ移行支援、並行運用、長期保守まで、関係者が共有すべき工程・役割・リスクを整理し、全体最適の観点で実施順序と責任を明確化する必要がある。特に富岳との互換性確認、データ移行方式、運用体制整備など複雑な要素を伴うため、導入計画の早期策定は、移行の停滞防止とシステム安定稼働に大きく寄与する。

上記理由から、「富岳NEXT」ではシステムだけではなく、施設インフラなども含めた総合的なシステム導入計画を策定することとする。導入の詳細計画（工程・体制・建屋調整・ユーザ移行計画等）は、詳細設計以降で策定することとする。以下に、詳細設計以降の検討項目を示す。

- 建屋建設・施設インフラ調整
- 電源・空調工事計画
- ケーブル配線計画
- 工事工程とシステム搬入工程の同期
- システム構築・導入工程
- 機器搬入計画
- システム構築計画
- 性能・運用試験の計画
- ユーザ移行支援計画
- アプリケーション移植・性能確認の支援
- ワークフロー移行ガイドライン
- チュートリアル・相談窓口・技術支援体制
- 並行運用および切り替え計画
- 富岳と「富岳 NEXT」の並行運用期間の定義
- データ移行方式・スケジューラ設定差異吸収の詳細
- 並行期間における性能検証・互換性試験

3.6.11 長期運用計画

「富岳NEXT」において、長期にわたる安定運用と「富岳NEXTNEXT」に向けたスムーズな移行を実現するための長期運用計画を策定する。この計画は「富岳NEXT」のライフサイクル全体を通じた継続性・信頼性・利用価値の最大化を目的とする。また、「富岳NEXTNEXT」へ向けた将来システムへの移行を円滑に行うため、計画段階から次期システムを見据えた運用方針と移行方式を組み込む。

長期運用計画の詳細は詳細設計以降にて検討する。以下に検討項目を挙げておく。

3.6.11.1 長期運用計画の策定方針

「富岳NEXT」の運用は、稼働開始から次期システムへの更新まで5年以上に及ぶ長期運用を想定する必要がある。そのため、単に現行システムの安定稼働だけを目的とするものではなく、「富岳NEXTNEXT」への移行を前提とした継続的な計画が必要となる。

3.6.11.1.1 並行運用期間の設計

「富岳NEXTNEXT」への切り替えにおいては、複数年度にわたる並行運用が必要となる。並行運用では、以下の要素を計画時点から明確化する：

- 課題・プロジェクト単位での段階的移行スケジュール
- データ移行方式（高速データ移行、差分同期、外部ストレージ連携等）
- アプリケーションビルド／最適化支援（GPU 世代差異等）
- 両システムでの WLM 共存戦略（キュー分離、利用枠管理）
-

3.6.11.1.2 ハードウェアの長期保守と更新

長期間の稼働を前提とし、以下の項目の事前検討を行う。

- 保守契約期間と更新タイミング
- 交換部材の確保、供給終了リスク管理
- GPU／ネットワーク／ストレージ等の世代更新時の運用影響評価

3.6.11.1.3 ソフトウェア・ミドルウェア更新計画

長期運用では更新頻度の高いコンポーネントについても体系的管理が必要である：

- OS の長期サポート（LTS）周期に合わせた更新計画
- WLM（Slurm/PBS/Flux 等）および Kubernetes のバージョン更新方針

以 上

付録 A:

1. アプリケーションワーキング サブワーキング報告書

1.1SubWG1 (生命科学分野)

1.1.1 目的と体制

SubWG１は、生命科学分野における代表的なアプリケーションを対象として、富岳NEXTに向けたアプリケーション特性の整理、ベンチマーク評価に向けた準備、および分野コミュニティとの情報共有を目的として活動を行った。

1.1.2 早期評価用アプリケーションの選定

生命科学分野からは、GENESIS（https://mdgenesis.org/）と UT-Heart（https://ut-heart.com/jp/index.html）を選定した。

GENESISはマルチスケール（全原子・粗視化・QM/MM）モデルに対応した分子動力学（MD）シミュレーションパッケージであり、MD計算に必要な各種機能および解析ツールから構成されるアプリケーションである。本アプリケーションは「京」開発時より重要な立ち位置を占め、「富岳」においてはターゲットアプリケーションとして採択されている。また、本アプリケーションの大部分はFortranで記述されているが、コードの一部はすでにCUDAで記述されており、GPUによる計算にも対応する。

本アプリケーションは、以下のような特性を有する。

- マルチスケールモデルに対応した MD 計算
- MPI と OpenMP によるハイブリッド並列化
- 「富岳」における最適化
- 一部計算は GPGPU による計算に対応
- 「京」および「富岳」においての最適化経験を有する開発者が在籍

これらの特性から、CPU・GPU を含む次世代計算機アーキテクチャ評価に対して有用なアプリケーションであると判断し、早期評価用アプリケーションとして位置付けた。

UT-Heartは心臓のマルチスケール・マルチフィジックスシミュレーションソフトウェアである。GENESISと同様に、本アプリケーションは「京」および「富岳」の開発時から関わりの深いアプリケーションであり、富岳において最適化されている。また、本アプケーションもGENESISと同様の特性を有することから早期評価用アプリケーションとして選定した。

1.1.3 ベンチマーク評価に向けた準備と提出

GENESISについてはオープンソースソフトウェアであるため、ベンチマーク評価に必要なソースコードの取得先をベンチマークWGに提示した。また、ベンダによる詳細な性能評価および将来的な最適化検討を目的として、NVIDIAおよび富士通にも対しても同様のものを提示した。

UT-Heartに関しては、理研-富士通-NVIDIAの３社間での契約締結を進めている。

1.1.4 SubWG 検討会議およびコミュニティ連携

SubWG 内および生命科学分野のアプリケーション開発者とは、主にオンラインミーティング、メール、Slackを用いて情報を交換した。SubWG内の会議においては以下の事項について継続的に議論を行なった。

- 早期評価用アプリケーション選定方針
- ベンチマーク WG からの要請への対応
- 分野内外のアプリケーション候補の整理

生命科学分野のアプリケーション開発者との間では、

- 本プロジェクトの目的・要望・進捗状況
- 開発者側からの意見・要望

- ベンチマーク WG との連携方法

について議論を行なった。とりわけGENESISの開発者からは、ベンチマーク対象として全原子モデルに加え、粗視化モデルおよびQM/MMモデルを含む広範なベンチマークテストの実施が要望された。また、プロジェクトのスケジュールおよびマイルストーンの提示についても強く求められた。

1.1.5 分野サーベイへの対応

2回実施された分野サーベイについては、SubWG内で対応した。

また、生命科学分野におけるAI利用アプリケーションの普及状況を把握するとともに、今後AIを活用したアプリケーションを対象とした性能評価を実施する可能性を見据え、SubWGおよびアプリケーション開発者との議論を踏まえてアプリケーションのリストアップを行った。リストを以下に示す。

プログラム名	**目的**
AlphaFold2	Structure Prediction
RoseTTAFold	Structure Prediction
RFdiffusion	Structure Generation
PhysNet	Machine Learning Potential for MD
Boltzmann Generators	Conformational Sampling
DeepEMhancer	Enhancing cryo-EM Density Maps
DeepFRET	Automatic analysis of single-molecule FRET data
ESM	Multimodal protein generative model
BioEmu	Sequence -> approximated equilibrium distribution of structures
DeePMD-kit	deep learning-based models of interatomic potential energy and force field
Boltz2	Structure prediction、affinity prediction
ChemTSv2	Molecule design
kMOL	Property prediction
INGOR	Gene network estimation
ABLang2	antibody-specific language model
BindCraft	antibody and peptide design
AFsample2	protein sequence -> generate multiconformations
Subsampled AF2	protein sequence -> generate multiconformations
BoltzGen	binder design
OpenFold3	structure prediction
ChemGLaM	PLM/CLM-based multimodal CPI prediction

1.1.6 今年度の成果と残された課題

GENESISを早期評価用アプリケーションとして選定し、評価準備を完了できたことは、SubWG1における主要な成果である。一方で、GENESISは計算の一部のみがGPUに対応しており、完全なGPU-residentアプリケーションではなく、計算の大部分をCPUに依存している。このため、GENESISのGPU-resident化を進めるとともに、GENESIS以外のGPU-residentアプリケーションを評価用として追加する必要があると考えている。また、ベンチマーク対象についても、

全原子モデルに加え、粗視化モデルおよびQM/MMモデルに対して実施する方針を明確にする必要がある。
UT-Heartについては、理研・富士通・NVIDIAの三者間契約に時間を要し、評価準備まで完了できなかった点が課題である。

1.1.7 来年度（詳細設計フェーズ）に向けた重要項目

GENESISについては、開発者と連携しつつ、詳細設計フェーズに向けてベンチマークフレームへアプリケーションをどのように組み込むかに関する検討を進めることが重要であると考えている。同時に、富士通およびNVIDIA社とアプリケーションの最適化方針について意見交換を行い、議論を深化させていく。また、GENESIS以外のGPU-residentなアプリケーション（例：GROMACS、OpenMM）についても、評価用アプリケーションとして利用可能か検討を進める。
UT-Heartについては、早期の契約締結を目指し、締結後はGENESISと同様に速やかに詳細設計フェーズに向けた準備を進める。

1.2SubWG2 (新物質・エネルギー分野)

1.2.1 目的と体制

SubWG2 の主な目的は、エネルギー・材料研究コミュニティを FugakuNEXT に向けて準備することであった。活動は次のようにまとめられる。

1. 分野における応用特性の調査
2. ベンチマークパラメータの設定
3. 科学コミュニティへの情報共有

材料科学分野は計算手法が極めて多様である。そこで、FugakuNEXT の強みを活かせる応用を選定し、コデザイン活動のために分野全体を広く把握することを目指した。

1.2.2 早期評価用アプリケーションの選定

以下の2つのアプリケーションを早期評価対象として選定した。

1. SALMON −光と物質の相互作用による電子動力学と光学応答を、時間依存密度汎関数理論（TDDFT）に基づき計算するソフトウェアである。Fugaku における実績と GPU への移植完了を踏まえ選定した。行列の繰り返し適用からなるステンシル型コードであり、メモリ帯域幅・レイテンシ要求が高い計算の代表例である。
2. mVMC −多変数変分モンテカルロ法による量子格子模型の数値計算ソフトウェアである。国内での広範な利用実績と HPCI 経由のアクセス可能性が選定の主な理由である。モンテカルロアルゴリズムのスケーラビリティと、稠密線形代数演算の多用という特性から、FugakuNEXT の浮動小数点演算性能評価に適している。

1.2.3 代替アプリケーションの検討

高度情報科学技術研究機構が提供するHPCIリソース用プリインストールソフトウェアセット（https://www.hpci-office.jp/for_users/appli_software）について、代替アプリケーションの候補として可能性を検討し、各アプリケーションの適合性を分析した。

- ABINIT-MP − 優れた代替候補だが、人的リソース不足のため選定できなかった。コード特性は材料分野より生命科学分野に近い。
- AkaiKKR − 同様の理由で選定できなかった。
- HΦ − 代わりに mVMC を選択。両コードは関連が深いため、mVMC の改善が HΦにも役立つ。
- LAMMPS − 国産コードではないから選定できなかった、材料科学における機械学習ポテンシャルの文脈で検討すべきコードである。
- NTChem − 利用者数が少ない。
- OpenMX ― 優れた代替候補だが、人的リソース不足のため選定できなかった。
- PHASE/0 − 利用者数が少ない。
- Phonopy − 本質的な計算コストは外部アプリケーションに依存する。

- Quantum ESPRESSO – 国産コードではなく、NVIDIA による性能評価も存在する。
- SMASH – 現在アクティブな開発が行われていない。

今後の検討対象として、ABINIT-MP、AkaiKKR、HΦ、OpenMXなどが推奨される。
NVIDIAのアプリケーションリストも代理候補として検討したが、材料関連はQuantum ESPRESSOのみで、SALMONは特有のカーネルを持つため代理として不適切と判断した。
本領域で頻出する計算性能制約型アプリケーションも調査の対象とした。稠密線形代数に制約される手法などが代表的である。

1.2.4 計算性能制約型・AI アプリケーション

計算性能制約型アプリケーションについても検討した。本領域ではこれらが頻出しており、特に稠密線形代数に制約される手法が代表的だ。ただし、実用上のピーク性能達成の困難さや低次スケーリングアルゴリズムとの競合などの課題がある。計算性能に制約されるか否かは、問題定義や数値表現などによっても変わる。代表的な例として、基底状態DFT計算 (10.1145/3295500.3357157)、励起状態計算 (arXiv:2509.23018)、高精度波動関数法 (10.1109/SC41406.2024.00015)、テンソルネットワーク法 (10.1021/acs.jctc.4c00903)、厳密対角化格子シミュレーション (10。1016/j.cpc.2024.109093) が挙げられる。大半では倍精度演算が必要だが、波動関数法については単精度演算で十分である。
AIアプリケーションについても検討した。材料分野における最も重要なユースケースは、機械学習原子間ポテンシャル（MILP）である。MILPは利用が拡大しており、スーパーコンピュータでピーク性能の大きな割合を達成するなど、高い計算要件を持つことがある（10.1016/j.cpc.2020.107624）。ニューラルネットワークアーキテクチャ、化学環境記述子、対称性を強制する追加手順の有無などに基づき、MILPには多様なバリエーションが存在する。最も普及しているのはMACEだが、より高性能な力場も登場しつつある（https://matbench-discovery.materialsproject.org/）。精度面では、MACE同等以上の品質と同一力場のバリエーションに焦点を当てる。例えば、eSEN（低精度使用を言及）、NequIP（単精度・倍精度混在）、SevenNet（単精度）、ORB（単精度）、DPA3（単精度・倍精度混在）、MACE（倍精度、論文で正当化）などである。原子間ポテンシャル以外では、拡散モデルなどの手法を用いた新規分子・材料の生成AIモデルが注目すべき新興アプリケーションである。

1.2.5 ベンチマーク評価に向けた準備と提出

早期評価対象アプリケーションの両方に対し、ベンチマーク実行用の入力ファイルおよびスクリプトを作成した。

1. SALMON については、FS2020 のプロジェクトのベンチマークが存在したが、今後の研究に必要な計算パターンとは異なると判断した。具体的には、既存のベンチマークは大規模な「k 点」サンプリングに依存しており、これは多数の小規模問題からなる Embarrassingly parallel な構造をもたらす。しかしながら、この手法はバルク系にのみ適用可能である。我々はその代わりに、スーパーセルサイズを増大させることに基づく新たなベンチマークを構築した。これは将来の科学的問題により適合したアプローチである。
2. mVMC については、開発者との議論に基づきベンチマークを用意した。異なるサイズの入力を生成するスクリプトも用意した。

これらのベンチマークをスーパーコンピュータFugakuおよびGrace Hopperシステム上で実行可能であることを確認した。SALMONについてはビルドシステム修正後に動作した（Github: be60c90）。さらに、SALMONのベンチマークをNVIDIAの開発者と共有し、FS2020で彼らが収集したデータとの違いについてもコメントを付け加えた。

1.2.6 SubWG 検討会議およびコミュニティ連携

SubWG2 は SALMON と mVMC の開発者コミュニティと会議を開催した。会議では、コード統合戦略について議論した。
SALMON開発者は、特定のブランチで開発を行い、メインGitHubリポジトリへのマージを基本とすることとした。CIはす

べてのマージリクエストをテストする。開発者は、コーディング規約を記載した開発者Wikiの共有を決定した。CUDAコードの寄与については、深層ルーチン（ハミルトニアン適用）向けかつオプションの場合に限り許可することとした。さらに、CXフレームワークに対し、当該変更や新コンパイラなどのテスト協力を要請した。なお、外部からのコード維持が従来困難であったこと、CUDAが主に物理研究者である開発者にとって難しいことについてもコメントがあった。
mVMCのライセンス上の制約から、NVIDIA統合の適切な戦略を立案することが重要である。当初はGPU向けPfaffianライブラリを作成するだけで十分と考えていたが、複数計算を同時実行することで性能向上を図るため、対応するドライバが必要である。コードはJuliaで書き換えられており、ジュリア版をより緩いライセンスで公開する可能性も検討中である。コードはメイン開発ブランチへのコミットが可能であり、開発者により進行中の各ブランチにリベースされる。

1.2.7 分野サーベイへの対応

SALMON と mVMC のプロジェクトに関わる開発者へ調査票を共有した。SALMON については両調査票への回答を、mVMC については第 2 調査票への回答を受領した。

1.2.8 今年度の成果と残された課題

本年度は、SALMONおよびmVMC開発者との連携によりコードデザイン活動に協力した。調査に加え、ベクトル幅に関する具体的な計算実験も実施した。さらに、FugakuNEXTで重要なアプリケーションを把握するため、分野全体の調査を行った。
残課題は以下の通り。

1. SALMON は開発者の FugakuNEXT への高い関心から、優先的なターゲットアプリケーションである。開発者と NVIDIA/Fujitsu 間の調整を継続し、可能な限り支援して GPU 性能を最大化する必要がある。
2. mVMC は GPL ライセンスの制約と GPU 版の欠如により難しい事例であるが、FugakuNEXT の浮動小数点性能を真に活用できる唯一の EEA である。開発者と積極的に連携し、ミニアプリケーションの開発と GPU 移植に人的リソースの投入が必要である。
3. 材料分野には多様な計算手法とソフトウェアがあるため、FugakuNEXT に向けたコード準備が進むようコミュニティへの働きかけを継続するとともに、FugakuNEXT が提供する巨大な計算能力を活かせる新手法の開発を促進すべきである。

mVMCの性能特性をFugaku上で性能ツールを用いて調査している。このデータを精緻化・要約した上で、FugakuNEXTにおけるピーク性能の達成度を把握する必要がある。

1.2.9 来年度（詳細設計フェーズ）に向けた重要項目

将来の取り組みは以下の通り。

1. SALMON および mVMC コードに対する特定 GPU 設計決定の影響を評価する実際の性能モデルと実験プラットフォームを構築する。
2. mVMC の GPU への移植を支援する。
3. SALMON コードをチューニングし、特定 GPU 設計への性能感度を把握する。
4. AI ワークロードベンチマークを提供し、プロジェクト領域「HPC アプリケーション WG」を支援する。

1.3SubWG3 (気象・気候分野)

1.3.1 目的と体制

SubWG3 は、気象・気候分野における代表的なアプリケーションを対象として、富岳 NEXT に向けたアプリケーション特性の整理、ベンチマーク評価に向けた準備、および分野コミュニティとの情報共有を目的として活動を行った。
基本設計フェーズにおいては、特にハードウェア設計に資する早期評価用アプリケーションの選定と、その評価に必要な情報・資材の整備に重点を置いた。

1.3.2 早期評価用アプリケーションの選定

気象・気候分野における早期評価用アプリケーションとして、SCALE-LETKF を選定した。

SCALE-LETKF は、領域気象モデル SCALE と局所アンサンブル変換カルマンフィルタ（LETKF）によるデータ同化システムを組み合わせたアプリケーションであり、数値予報における代表的な計算パターンを含んでいる。
本アプリケーションは、以下のような特性を有する。

- 大規模三次元格子計算と時間発展計算を含む数値モデル計算
- LETKF における行列演算を中心としたデータ同化計算
- MPI 通信とメモリアクセスが支配的となる計算構造
- 入出力データの大規模ファイル I/O

これらの特性から、CPU・GPU を含む次世代計算機アーキテクチャ評価に対して有用なアプリケーションであると判断し、早期評価用アプリケーションとして位置付けた。

1.3.3 ベンチマーク評価に向けた準備と提出

SCALE-LETKF について、ベンチマーク評価に必要な以下の資材を整備し、ベンチマーク WG に提出した。

- ソースコード一式
- ベンチマーク実行用の入力データ
- コンパイル手順、実行手順、実行結果確認方法をまとめたドキュメント

これらは、富岳および GPU を含む複数アーキテクチャ上での評価が可能となるよう整理したものである。
また、ベンダによる詳細な性能評価および将来的な最適化検討を目的として、NVIDIA および富士通に対して SCALE-LETKF のソースコードおよび関連データを提供した。

1.3.4 SubWG 検討会議およびコミュニティ連携

活動期間中に、SubWG3 として 計 9 回の検討会議を開催し、以下の事項について継続的に議論を行った。

- 早期評価用アプリケーション選定方針
- ベンチマーク WG からの要請への対応
- GPU 対応状況および今後の開発方針
- 分野内外のアプリケーション候補の整理

分野内外のアプリケーション候補に関しては、現在、4つのアプリケーションが将来的なベンチマークへの貢献に関心を示している。
加えて、2026 年 3 月 26 日に、気象・気候分野に向けた富岳 NEXT プロジェクトの状況共有と将来展望の議論を目的とした研究会の開催を企画した。
本研究会は、分野コミュニティとの情報共有および意見交換の場として位置付けられている。

1.3.5 分野サーベイへの対応

プロジェクト側から実施されたアプリケーション開発者向けサーベイに対し、気象・気候分野として回答を取りまとめた。

- 第 1 回サーベイ：10 件
- 第 2 回サーベイ：5 件

これらは分野全体としての貢献であり、SubWG3 はその取りまとめおよび提出を担った。

1.3.6 SIMD 長に対する影響評価

SIMD 長の違いがアプリケーション性能に与える影響を評価するため、SCALE（LETKF 部分を除く）について、富岳においてコンパイラオプションおよび実行時環境設定を変更することで、256 bit SVE と 512 bit SVE の実行時間比較を行った。
その結果、512 bit を用いた場合、256 bit 実行と比較して約 15% の高速化が確認された。
一方で、この性能差については、SIMD 長の違いのみで説明できるかどうかについては慎重な解釈が必要である。
SCALE の多くのカーネルはメモリ律速であるか、分岐を多く含む構造を持っており、単純に SIMD 幅の拡張のみで大幅な性能向上が得られるとは必ずしも直感的ではない。

実際には、SIMD 長の違いに加え、

- コンパイラ最適化の挙動の差
- ベクトル化の適用範囲の変化
- 命令スケジューリングやレジスタ利用の差

など、複合的な要因が影響している可能性がある。
本評価は限定的な条件下での比較であり、SIMD 長の影響を厳密に切り分けたものではないが、アーキテクチャ設計およびコンパイラ最適化戦略を検討する上での基礎的な情報として共有する。

1.3.7 今年度の成果と残された課題

基本設計フェーズにおいて、SCALE-LETKF を中心とした早期評価用アプリケーションの選定および評価準備を完了できたことは、SubWG3 の主要な成果である。一方で、今年度の実施内容に対して、以下の課題が明確になっている。

- LETKF 部分の GPU 化が未完了であり、GPU を活用した性能評価・最適化は今後の重要課題である。
- SCALE 部分については GPU 上での実行は可能であるものの、現時点では性能向上の余地が残されている。
- 各 SubWG から 1～2 本の評価アプリケーション提出が想定されていたが、気象・気候分野では結果として 1 本の提出に留まった。

1.3.8 来年度（詳細設計フェーズ）に向けた重要項目

来年度の詳細設計フェーズに向けて、SubWG3 としては以下の項目が重要であると考えている。

- LETKF 部分の GPU 化と計算構造の再整理
  LETKF 部分については、GPU 化が未完了であり、今後、GPU を前提とした実装および性能評価を進める必要がある。特に、行列演算を中心とした計算構造については、GPU に適したデータ配置やライブラリ活用を含めた見直しが重要である。
- SCALE 部分における GPU 上での性能改善
  SCALE 部分は GPU 上での実行は可能であるものの、現時点では十分な性能向上が得られていない。メモリアクセスやカーネルへの分解方法等の観点から、さらなる最適化余地が残されている。
- アルゴリズム要素のサロゲート化および再構成の検討
  数値予報・データ同化システム全体を構成する各要素について、計算コストの高い部分のサロゲート化、アルゴリズム構成の再整理といった観点からの検討が今後重要になると考えられる。これにより、従来型の計算だけでなく、AI・機械学習手法を導入する可能性についても検討対象とする。
- 低精度演算および行列演算エンジン活用の可能性検討
  LETKF をはじめとする行列演算を多用する計算においては、低精度演算の適用可能性、専用の行列演算エンジンの活用、といった観点からの検討も、将来的な性能向上や電力効率改善に寄与する可能性がある。詳細設計フェーズでは、数値精度要件との関係を整理しつつ、これらの適用可能性を評価する。
- 計算特性の異なる追加アプリケーションの選定と提出
  気象・気候分野には多様なアプリケーションが存在することを踏まえ、通信特性、メモリ律速性、行列演算比率などの観点から重要なアプリケーションを追加で選定し、ベンチマークとして提出することが重要である。これにより、富岳 NEXT のアーキテクチャ設計に対して、より多面的な入力を与えることが可能となる。

これらを通じて、詳細設計フェーズにおけるアーキテクチャ設計および最適化検討に対して、より広範かつ実態に即した入力を行うことを目指す。

1.4SubWG4 (地震・津波防災分野)

1.4.1 目的と体制

SubWG4は、地震・津波防災分野における代表的なアプリケーションを対象として、富岳 NEXT に向けたアプリケーシ

ョン特性の整理、ベンチマーク評価に向けた準備、富岳NEXTにおけるCPU-GPUの協調利用に向けたアプリケーション開発に関する調査を行った。基本設計フェーズにおいては、特にハードウェア設計に資する早期評価用アプリケーションの選定と、その評価に必要なプログラム・サンプル入力データの整備に重点を置いた。

1.4.2 早期評価用アプリケーションの選定、ベンチマーク評価に向けた準備と提出

地震・津波防災分野における早期評価用アプリケーションとして、E-Waveを選定した。
E-Waveは、陰解法の非構造格子有限要素法に基づく地震波動伝播計算アプリケーションであり、地震・津波防災分野における代表的な計算パターンを含んでいる。
本アプリケーションは、以下のような特性を有する。

- 非構造格子有限要素法において用いられるランダムアクセスを主体とした疎行列ベクトル積計算
- 共役勾配法ソルバにおけるメモリバンド幅律速となる計算
- 疎行列ベクトル積における隣接領域における MPI 通信、及び、共役勾配法ソルバにおける MPI_Allreduce 通信

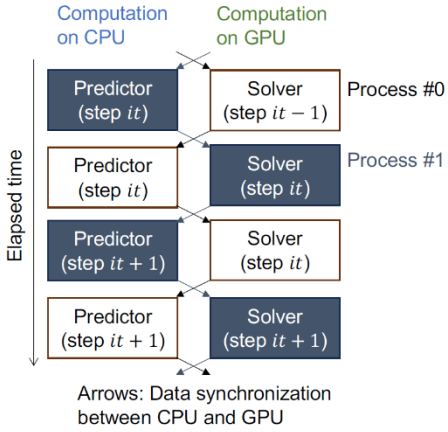
これらの特性から、次世代計算機アーキテクチャ評価に対して有用なアプリケーションであると判断し、早期評価用アプリケーションとして位置付けた。
E-Waveについて、ベンチマーク評価に必要な以下の資材を整備し、ベンチマークWGに提出した。

- ソースコード一式
- ベンチマーク実行用の入力データ
- コンパイル手順、実行手順、実行結果確認方法をまとめたドキュメント

これらは、富岳および NVIDIA GPUを含む複数アーキテクチャ上での評価が可能となるよう整理したものである。このコードが富岳NEXTプロジェクト内で活用できるよう、NDA・共同研究契約に基づく研究体制の整備に向けた検討を実施した。

1.4.3 CPU-GPU の協調利用に向けたアプリケーション開発に関する調査

現行のE-Waveを富岳NEXTが想定するCPUおよびGPUからなるheterogeneous architectureにおいて実行する際は、性能の高いGPUにおいて全計算を実行し、CPUはGPUの制御のみに用いられることとなり、CPUコアやCPUのメモリが活用されないこととなる。GPUリソースとともにCPUリソースを用いることで、実行時間および必要エネルギーの削減につながる可能性がある。そこでSubWG4においては、分野におけるCPU-GPUにおける協調計算例を調査し、それをFugakuNEXT Application Development Seminarにおいて紹介した(発表タイトル：Simultaneous use of CPU and GPU for fast and energy-efficient implicit wave simulation)。ここでは本分野におけるCPU-GPU協調計算の開発例として、リソースの同時活用により多数ケースの時刻歴シミュレーションにおける実行時間（time-to-solution）と消費エネルギー（energy-to-solution）を同時に削減する手法を紹介した[Ichimura et al. 2024]。本手法では、大容量なCPUメモリに格納された過去の求解結果から、CPU計算を用いて次ステップの初期解を高精度で推定することで、シミュレーション結果の精度を維持したまま反復法ソルバの反復回数を低減させ計算を高速化している。計算負荷の高いソルバ実行には高速なGPUを割り当て、2ケースを同時並列で処理することでCPUとGPUを同時に使うことで演算リソースを最適化しており、他の高性能計算手法との組み合わせにより、GH200環境においてGPU単体での従来手法比で8。7倍のスループット改善およびエネルギー使用量1/7という効率化を実現している。
これらのCPU-GPUの協調計算に関する調査結果に基づいてDeveloper Surveyのサーベイアンケートを回答した。



Tsuyoshi Ichimura、Kohei Fujita、Muneo Hori、Maddegedara Lalith、Jack Wells、Alan Gray、Ian Karlin、John Linford: Heterogeneous computing in a strongly-connected CPU-GPU environment: fast multiple time-evolution equation-based modeling accelerated using data-driven approach、Workshop on Accelerator Programming and Directives (WACCPD) 2024@SC24

1.5SubWG5 (ものづくり分野)

1.5.1 目的と体制

ものづくり分野においては、現象解明や試験の完全な代替えを目的とした第一原理的な高精度解析や、様々な拘束条件を考慮した多目的最適化に対して、HPC の適用が進んでいる。特に近年では、多種・多様な市場ニーズを満たすために製品開発期間の短縮が求められていることから、HPC を活用した設計プロセスの革新への期待が高まっている。SubWG5 は、ものづくり分野において用いられるアプリケーションを対象として、最新の動向やニーズを調査し、その結果を踏まえて、次世代計算基盤におけるターゲット問題および必要となる計算資源の整理を行うことを目的として活動を行った。

基本設計フェーズにおいては、特にハードウェア設計に資する早期評価用アプリケーションの選定と、その評価に必要な情報・資材の整備に重点を置いた。

1.5.2 早期評価用アプリケーションの選定

早期評価用アプリケーションは、ものづくり分野における主要課題である乱流解析から選定した。乱流解析では、メモリサイズ、メモリバンド幅、ノード間通信性能がアプリケーションの適用範囲を決めることが多く、GPU と DRAM 間のデータ転送や、DRAM 内におけるデータ移動を重視する次世代計算基盤のアーキテクチャによって大幅な性能向上が期待できるためである。この観点から、メモリアクセス特性の異なる 2つのアプリケーション、すなわち FrontFlow/blue（以下、FFB）および FFVHC-ACE を早期評価用アプリケーションとして選定した。

FFB は、非圧縮性・圧縮性流体の非定常流動を高精度に予測可能な Large Eddy Simulation（LES）に基づいた汎用流体解析コードである。形状適合性に優れた有限要素法を採用しており、ファン・ポンプ等の流体機械や複雑形状周りの非定常乱流解析、ならびに流れから発生する騒音の予測が可能である。また、ベクトル計算機およびスカラー型超並列計算機上で高速動作するように最適化されており、自動化された最適領域分割・統合処理を実装した領域分割法によって、大規模超並列計算に対応する。実際に、「京」、「富岳」をはじめとする大規模並列計算機での実績を有し、その成果を記載した論文はSC20（International Conference for High Performance Computing、Networking、Storage、and Analysis）においてゴードン・ベル賞の最終候補論文に選出されている。

一方、FFVHC-ACE は、航空機機体などの複雑形状周りの高忠実な圧縮性流体 LES 解析を実現する流体解析コードであり、以下の 3つのキー技術を特徴とする。

1) 複雑形状に対して完全自動格子生成を可能とする階層型等間隔直交格子法と独自の埋め込み境界法
2) 高レイノルズ数流れの LES 解析を可能とする壁面モデル LES
3) 高忠実な圧縮性流体 LES 解析を実現する数値計算スキーム（KEEP スキーム）

FFB と FFVHC-ACE の大きな違いは、流れ場の情報を格納する内部データ構造にある。FFB は非構造データを用いるため、主要な演算部ではリストベクトルを多用する計算構造となる。一方、FFVHC-ACE は構造格子データを用いており、主にステンシル計算が支配的となる。これら 2つのアプリケーションは、メモリアクセス特性および計算パターンが大きく異なることから、次世代計算機アーキテクチャ評価に対して相補的なベンチマークとなると判断した。

1.5.3 ベンチマーク評価に向けた準備と提出

FFBについて、ベンチマーク評価に必要な以下の資材を整備し、ベンチマーク WG に提出した。FFVHC-ACEに関しては下記準備を進めたが、ライセンスおよび公開性に関して調整中である。

- ソースコード一式
- ベンチマーク実行用の入力データ
- コンパイル手順、実行手順、実行結果確認方法をまとめたドキュメント

これらは、「富岳」および GPU を含む複数アーキテクチャ上での評価が可能となるよう整理したものである。

1.5.4 SubWG 検討会議およびコミュニティ連携

活動期間中に、SubWG5 として 3 回の検討会議を開催し、以下の事項について継続的に議論を行った。

- 早期評価用アプリケーション（FFB、FFVHC-ACE）の選定方針
- ベンチマーク WG からの要請への対応

1.5.5 分野サーベイへの対応

プロジェクト側から実施されたアプリケーション開発者向けサーベイに対し、ものづくり分野として回答を取りまとめた。これらは分野全体としての意見集約であり、SubWG5 はその整理および提出を担った。

1.5.6 今年度の成果と残された課題

基本設計フェーズにおいて、FFB および FFVHC-ACE を中心とした早期評価用アプリケーションの選定およびFFBのベンチマーク評価準備を完了できたことは、SubWG5 の主要な成果である。一方で、今年度の実施内容に対して、以下の課題が明確になっている。

- FFB は GPU 上での実行は可能であるものの、今後詳細なプロファイルの取得が必要であり、現時点では性能向上の余地が残されている。
- FFVHC-ACE は GPU への対応が必要である。
- ベンダによる詳細な性能評価および将来的な最適化検討を進めるためには、関係ベンダに対してソースコードおよび関連データを提供するためのリポジトリ管理やライセンスの整備・明確化が必要である。

1.5.7 来年度（詳細設計フェーズ）に向けた重要項目

来年度の詳細設計フェーズに向けて、SubWG5 としては以下の項目が重要であると考えている。

- ベンダによる性能評価を円滑に進めるための環境整備
  FFB および FFVHC-ACE について、ベンダによる詳細な性能評価および最適化検討を円滑に進めるため、ソースコードおよび関連データの提供体制の整備が最優先の課題である。具体的には、すでに合意した方針に基づいたリポジトリの構築および運用、ライセンス条件の明確化、配布手順の整備などを進め、評価環境の早期立ち上げを可能とする必要がある。
- GPU を前提とした実装および性能最適化の推進
  ベンダによる性能評価結果を踏まえ、主要計算部分を中心にチューニングを実施し、主要な演算部としてGPU を前提とした性能最適化を段階的に進めることが重要である。

これらを通じて、ベンダ側からの評価結果を一方的に受け取るだけでなく、アプリケーション開発者の立場から計算特性

や実運用上の要請を整理し、ハードウェア設計に対するフィードバックを行うことで、詳細設計フェーズにおけるアーキテクチャ設計および最適化検討に対して、より広範かつ実態に即した入力を行うことを目指す。なお、このような協調活動を効率的に進めるためには、ハードウェアの設計情報の開示が不可欠であることを付記する。

1.6SubWG6 (基礎科学分野)

1.6.1 目的と体制

SubWG6は、基礎科学分野における代表的なアプリケーションを対象として、富岳NEXTに向けたアプリケーション特性の整理、ベンチマーク評価に向けた準備、および分野コミュニティとの情報共有を目的として活動を行った。

1.6.2 早期評価用アプリケーションの選定

基礎科学分野からは、LQCD-DWF-HMC（https://github.com/i-kanamori/LQCD-DWF-HMC） と HDTurbulence（https://github.com/cfcanaoj/MHDTurbulence）を選定した。

LQCD-DWF-HMC は素粒子・原子核分野からの選定で、格子QCD（Lattice QuantumChromodynamics; Lattice QCD または LQCD）のシミュレーションコードで、素粒子であるクォークとグルーオンの物理を対象としたアプリケーションである。格子QCDシミュレーションは、並列計算機の発展と軌を一にして発展してきており、HPCI課題の中でも大きな割合を占めている（「富岳」一般課題の採択課題に配分された計算資源中、令和６年度Ａ期は18%、令和７年度Ａ期は14%）。選定コードでは、クォークの記述にQCDの重要な性質であるカイラル対称性の扱いにすぐれたドメインウォール型（Domain-Wall Fermion、 DWF）を用い、ハイブリッドモンテカルロ法（Hybrid Monte Carlo; HMC）でグルーオン場の配位を生成する。計算の主要部は偏微分方程式を反復法で解くソルバで、この部分を抜き出したものもカーネルアプリとして提供された。使用言語は C++ で、アーキテクチャ固有の最適化を除き C++11の範囲で記述している。

本コードは、以下のような特性を有する。

- 複素数の取り扱いが必要である
- 構造格子上でのステンシル計算に伴う高頻度の隣接通信が生じる
- 並列化は MPI と OpenMP によるハイブリッド
- 反復法に伴う global reduction も高頻度である
- メモリバンド幅・通信律速である
- FP32 との混合精度アルゴリズムが使われている
- 実行時に初期条件を与えるファイルを別途用意する必要はない
- GPU 化は OpenMP（カーネルアプリについては、OpenMP の他に CUDA 版も提供）

これらの特性から、GPUを含む次世代計算機アーキテクチャの評価に対して有用なアプリであると位置づけた。ライセンスは、もとになったコードセット（Bridge++; https://bridge.kek.jp/Lattice-code/ ）から GPLv3 を継承しているが、カーネル部分については MIT ライセンスのものも提供準備中である。

MHDTurbulence は宇宙分野からの選定で、三次元磁気流体力学（Magnetohydrodynamics; MHD）方程式を解く乱流シミュレーションコードであり、宇宙プラズマや星間媒質を対象とする基礎的かつ汎用的なアプリケーションである。本コードは有限体積法（Finite Volume Method）に基づき圧縮性流体方程式を解くものであり、太陽、恒星、超新星爆発などの天体物理現象、さらには銀河形成・進化や降着円盤シミュレーション等、より広範な天体物理問題への応用が可能である。

本コードは Fortran90 を用いており、以下のような特徴を有する。

- 三次元一様格子上での大規模有限差分計算
- 時間発展型陽的スキームによる反復更新
- MPI による領域分割並列
- OpenMP および OpenACC による GPU オフロード実装

- 実行時に初期条件を与えるファイルは不要

これらの計算特性は、メモリ帯域、通信性能、演算性能（特にベクトル演算性能）への依存度が高く、GPU 加速器を含む次世代アーキテクチャの評価に対して有用であると判断した。また、基礎物理に立脚しつつも計算構造が比較的単純であり、性能評価や最適化検討を進めやすいアプリケーションであることから、早期評価用アプリケーションとして位置付けた。ライセンスは BSD 3-Clause である。

なお、どちらのコードも利用する言語仕様については保守的でかつ依存ライブラリも少ないため、可搬性が高いものになっている。

1.6.3 ベンチマーク評価に向けた準備と提出

LQCD-DWF-HMC については、ベンチマーク評価に必要な以下の資材を整備し、ベンチマークWGに提出した。

- ソースコード一式
- ベンチマーク実行用の入力データ
- コンパイル手順、実行手順、実行結果確認方法をまとめたドキュメント

これらは、富岳とNVIDIAのGPUで評価が可能となるように整理してある。また、ベンダによる詳細な性能評価および将来的な最適化検討を目的として、NVIDIAおよび富士通に対しても同様なものを提供し、個別の問い合わせに対応した。また、富岳を用いた通信性能の評価にもソースコードとパラメータ、実行環境に関わる情報を提供している。

SIMD長による性能変化についてはベンチマーク結果を提供した。

このアプリケーションのソースコードはフルアプリ版とカーネル版の2種類ある。フルアプリ版については、当初は tar ball による提供だったが最終的に GPLv3 ライセンスのもと、github にて公開した。カーネル版については報告書執筆時で tar ball のままだが、開発者の同意を取り、ライセンスをMITに変更ものを github にて公開する準備を進めている。なおtar ball のカーネル版は、benchkit に取り込み済みである。

MHDTurbulence については、ベンチマーク評価に必要な以下の資材を整備し、ベンチマーク WG に提出した。

- ソースコード一式
- コンパイル手順、実行手順、実行結果確認方法をまとめたドキュメント

これらは、CPU および GPU を含む複数アーキテクチャ上での評価が可能となるよう整理したものである。当初は GPU 版のみが整備されていたが、多様なアーキテクチャ上でのベンチマーク実行を可能とするため、本活動の一環として CPU 版も新たに整備した。

1.6.4 SubWG 検討会議およびコミュニティ連携

SubWG 内では、主にオンラインとメールで情報を交換した。

基礎科学分野のコミュニティ全体に対しては、計算基礎科学連携機構（JICFuS）主催で2025年9月に開催された第25回HPC-Phys勉強会で富岳NEXTにとアプリの動向についての情報共有がなされた。

基礎科学内の素粒子・原子核分野では、調査研究に関わった開発者を中心に、ベンチマークに関する情報の提供を随時行っている。LQCDの範囲内でもクォークの扱いでDWFとは異なる型を用いているグループや、AI加速のアルゴリズムを調査している研究者とも連絡をとっている。

宇宙分野の富岳利用者を中心に、ベンチマークに関する情報共有を行っている。コミュニティ内では、適合格子法（AMR）や反復法を含む、より複雑なアプリケーションの GPU オフロードに対する関心が高い。早期評価の次段階においては、こうした計算特性の異なるコードも対象として扱うことが望まれている。

1.6.5 分野サーベイへの対応

2回行われた分野サーベイについては SubWG内で対応した。

1.6.6 今年度の成果と残された課題

LQCD-DWF-HMC 並びに HDTurbulence を早期評価用アプリケーションとして選定したこと、とくに前者について基本設計フェーズにおける評価につなげたことは、SubWG6の主要な成果である。一方で、以下の課題が明確になっ

た。

- LQCD-DWF-HMC について、ベンダが求めるライセンス（GPL でないこと）との不一致。カーネル部分については GPL ではない形のものを今後提供予定である。
- ソースの提供方法に関して、LQCD-DWF-HMC が公開の形での提供なっていなかった。フルアプリについては 2025 年 12 月に github にて公開したが、カーネルアプリの公開が遅れている。

1.6.7 来年度（詳細設計フェーズ）に向けた重要項目

LQCD-DWF-HMC は、カーネルアプリの非GPLライセンス版の公開を急ぐとともに、G P U向けの通信の最適化をすすめる。また、NVIDIA が GPU 向けに開発している関連ライブラリ QUDA の活用をすすめる。元アプリ（Bridge++）で開発が進んでいる半精度演算を利用したアルゴリズムも取り込んでいく。また、本アプリではポッドサイズが性能に大きな影響を与える可能性が示唆されている（RFPによると12%程度）一方、半精度演算を利用したQUDAはその差を縮小することも示唆されている。詳細設計でその影響について詳細な調査を行うとともに、ハードウェア設計にフィードバックする必要がある。

現状、MHDTurbulence は OpenMP（CPU）、OpenMP（GPU）、OpenACC 版と複数の実装に分かれている。今後は Kokkos を導入することで、さまざまな計算環境において統一的な実装基盤を確立し、より容易かつ公平な性能比較を可能とすることを目指す。詳細設計フェーズに向けて、Kokkos への移植を進め、アーキテクチャ依存部分の整理と移植性向上を図る。

1.7SubWG7 (スマートシティ分野)

1.7.1 目的と体制

SubWG7 はスマートシティ領域の代表的アプリケーションを対象に、富岳 NEXT に向けたアプリケーション特性の整理、ベンチマーク準備を目的に活動した。基本設計フェーズでは、次世代計算機の評価に資する早期評価用アプリケーションの候補選定と、ベンチマークを実行できる状態にするための最低限の準備に重点を置いた。対象は、都市スケールでの電波伝播・電磁界解析、通信方式（ミリ波・テラヘルツ）評価、電力・再生可能エネルギー挙動（例：窓型ペロブスカイト太陽電池を含む ZEC）、スマートグリッド、EV の充放電挙動、車両・人流・エージェント挙動など、マルチフィジックス／マルチエージェントを含むシミュレーション群である。ただし現時点では完全な統合シミュレータは未完成であり、段階的に要素ごとの実行環境とベンチマークを整備している段階である。

1.7.2 早期評価用アプリケーションの選定

スマートシティ分野における早期評価用アプリケーションとして、NVIDIA Sionna と Agentic AI を組み込んだ SUMO を選定した。Sionna は GPU を活用したレイトレーシングベースの 3 次元無線電場伝搬／リンクレベルシミュレーション機能を備え、Agentic AI を組み込んだ SUMO は車両・人流・エージェント意思決定の評価基盤を提供する。現時点では両者が単体で動作し、ベンチマーク取得が可能な状態にある。

本アプリケーションは、以下のような特性を有する。

- 大規模三次元レイトレーシング等を含む空間計算（高解像度空間データに対する多数の光線／経路探索）。
- 行列演算やニューラルネットワーク推論を中心とする計算（ビームフォーミング、チャネル行列生成、Agentic AI の推論等）。
- 多数エージェントのイベント駆動に伴う分岐の多さと不規則なメモリアクセス（条件分岐・動的データ構造が支配的）。

これらの特性から、CPU・GPU を含む次世代計算機アーキテクチャの性能評価に対して有用であると判断し、早期評価用アプリケーションとして位置付けた。

1.7.3 ベンチマーク評価に向けた準備と提出

SUMO、Sionna両者について、ベンチマーク評価に必要な以下の資材を整備し、ベンチマーク WG に提出した。

- ソースコード一式

- ベンチマーク実行用の入力データ
- コンパイル手順、実行手順、実行結果確認方法をまとめたドキュメント

これらは、富岳アーキテクチャ上での評価が可能となるよう整理したものである。

1.7.4 SubWG 検討会議およびコミュニティ連携

活動期間中に、SubWG7内部で 計 2回の検討会議、ベンチマークWGとの検討会議を開催し、以下の事項について継続的に議論を行った。

- 早期評価用アプリケーション選定
- ベンチマーク WG へのアプリケーションの説明と開発段階の共有
- ベンチマーク WG からの要請への対応
- GPU 対応状況および今後の開発方針

1.7.5 分野サーベイへの対応

プロジェクト側から実施されたアプリケーション開発者向けサーベイに対し、スマートシティ分野として回答を取りまとめた。

- 第 1 回サーベイ：1 件
- 第 2 回サーベイ：1 件

1.7.6 今年度の成果と残された課題

基本設計フェーズにおいて、NVIDIA Sionna および Agentic AI を組み込んだ SUMO を早期評価用アプリケーションとして選定し、それぞれ単体で動作可能な状態とし、ベンチマーク取得が可能な環境を整備したことが SubWG7 の主要な成果である。また、富岳アーキテクチャ上での実行を想定したビルド・実行手順の整理を行い、評価基盤の初期整備を完了した。

一方で、以下の課題が明確となっている。

- Sionna、SUMO、今後導入予定の電磁界・電力系シミュレータ等を含めた統合シミュレータは未完成であり、現状は要素単体での評価に留まっている。
- GPU 前提の最適化コード整備や MPI によるエリア分割並列化は未実装または限定的であり、大規模スケーラビリティ評価が未実施である。
- Singularity 環境を中心に開発を進めているが、Spack 対応、富士通コンパイラでのビルド検証、Benchpack／BenchKit への適合など、富岳 NEXT を前提とした標準的運用環境への整備が未完了である。

1.7.7 来年度（詳細設計フェーズ）に向けた重要項目

詳細設計フェーズに向けて、SubWG7 としては以下の項目が重要である。

- シミュレータの統合化
  Sionna、Agentic AI 組込 SUMO、および今後導入予定の電磁界・電力系モデルを連携させ、都市スケールでの統合評価が可能な構成を確立する。
- MPI 対応
  都市空間の領域分割に基づく MPI 並列化を実装し、スケーラビリティ評価を可能とする。
- ビルド・実行環境の標準化
  現在の Singularity ベースの開発環境に加え、Spack への対応を行うとともに、富士通コンパイラでのビルドおよび動作検証を実施する。
- Benchpack／BenchKit 対応
  ベンチマーク WG の運用基盤に適合する形でアプリケーションを整理し、標準的な評価フレームワークに組み込める形へ整備する。

これらを通じて、詳細設計フェーズにおいて、統合スマートシティシミュレーションの実行基盤を確立し、アーキテクチャ設計および性能評価に対して実効的な入力を行うことを目指す。

1.8SubWG8 (科学技術計算・機械学習アルゴリズム分野)

1.8.1 目的と体制

SubWG8は、数値シミュレーションと深層学習における基本的な演算ライブラリを対象として、富岳NEXTでのアプリケーションに利用される場合の特性の整理、ベンチマーク評価に向けたライブラリの性能評価のためのミニアプリケーションの設定とデータセットの準備、および分野コミュニティとの情報共有を目的として活動を行った。

1.8.2 早期評価用アプリケーションの選定

数値シミュレーションと深層学習分野からは、PTESc（https://petsc.org)と Pytorch（https://pytorch.com）を選定した。これらはBLAS や LAPACK に代表される基盤数値ライブラリと実アプリケーションの中間に位置する並列演算環境での数値演算ライブラリ集である。

PETSc は偏微分方程式によって記述される数理モデルの離散化から得られる疎行列からなる線形／非線形の連立方程式系をデータ分散に基づき並列計算環境で求解を行うための数値演算ライブラリでありC言語によって記述され、アルゴンヌ国立研究所により開発が継続されている。疎行列は非構造データの格納形式であるCSR形式で保持され、大規模な疎行列の連立一次方程式の解法であるクリロフ部分空間法といくつかの高性能な前処理が実装されている。ここで基本となる数値演算は疎行列とベクトルの積を計算するSpMVである。このライブラリは次の特徴を持つ。

- CPU と GPU が混在したヘテロジニアスな並列環境での MPI プロセスの実行
- 行列データ形式への抽象化されたインターフェースによりデータ分散操作と線形／非線形ソルバーアルゴリズムの分離
- C 言語によるインターフェースだけでなく、Fortran および Python で記述されるアプリケーションへのインターフェース

これらの特性からGPUを含む次世代計算機アーキテクチャの評価に対して有用な数値ライブラリであると位置づけた。

Pytorchは深層学習用の基本的数値ライブラリでありPython言語によるインターフェースを提供している。大規模言語モデル(LLM)はTransformerアーキテクチャを中心に発展してきたがその基盤を浮動小数点演算とデータ分散および並列実行のための通信から支えている。このライブラリは次の特徴を持つ。

- CPU と GPU が混在したヘテロジニアスな並列環境で実行され MPI による分散実行に加えて GPU 間の直接データ転送をサポートしている。
- 演算粒度が高くデータ通信に比較して演算が支配的であり、かつ演算密度も高いため、最新の GPU の演算器を十分に活用している。
- 学習と推論のフェーズがあるが、双方で混合精度演算による高速化が進んでいる。

これらの特性からPETScライブラリが扱う疎行列データを相補する密データを扱う数値演算を用いてGPUを含む次世代計算機アーキテクチャの評価するために有用な数値ライブラリであると位置づけた。

1.8.3 ベンチマーク評価に向けた準備と提出

PETScとPytorchともに数値演算ライブラリであるため、その性能評価にはライブラリを利用するためのアプリケーションとデータセットが必要である。PETScにおいては有限要素法から生成される疎行列データをMatrixMarket形式で保存しているアスキーファイルを読み、PETSc内部の分散保持された疎行列データに格納してクリロフ部分空間法により数値解を求めるC言語によるプログラムを作成した。クリロフ部分空間法と前処理の選択および反復実行と収束判定のためのパラメータを設定するスクリプトにより実行される。Pytorchの評価にはMegatron-LM(https://github.com/NVIDIA/Megatron-LM/) によるLLMの学習を選定した。対象とするモデルにDenseモデルとMoEモデルの双方を選び分散学習設定のためのパラメータを準備した。

上記2種類のスクリプトとデータ取得先およびパラメータ設定からなるベンチマークセットをベンチマークWGに提出した。

1.8.4 SubWG 検討会議およびコミュニティ連携

活動期間中にSubWG8において計9回の検討会を開催し、以下の事項について Zoomでの会議に加え Slack の

チャンネルを含めて継続的に議論を行った。

- 早期評価用アプリケーション選定方針
- ベンチマーク WG からの要請への対応
- GPU 対応状況および今後の開発方針
- 他のアプリケーション WG での数値ライブラリの活用の促進について

PETScのコミュニティに直接コンタクトを取り、GPU環境での実行可能性と複数GPUを持つ計算ノードでの開発動向に関する情報を得た。

- GPU においてはハードウェアの特性から演算直後のデータの再利用を行う前処理は実装されていないこと
- マルチ GPU の実行をサポートする新しい分散環境のための通信ライブラリ PETScSF が開発版で実装されていること

1.8.5 分野サーベイへの対応

2回行われた分野サーベイについては SubWG内で対応した。特にPETScが扱う疎行列による線形／非線形問題はメモリ律速である。しかしながら、いくつかの前処理実装では演算密度を高められる可能性があるため、GPU環境でのメモリ帯域と演算速度のバランスについて検討した事項を回答した。

1.8.6 今年度の成果と残された課題

PETScを利用するミニアプリケーションとして、当初はFotran90によるオープンソースの弾性体シミュレーションコードを想定していた。しかしNVIDIAによるFortran コンパイラが、最新の言語仕様を全て実装していないため、PETScのFortranインターフェース部分をコンパイルすることができないことが判明した。このためC++言語による疎行列ソルバの実行ルーチンを記述し、疎行列を評価アプリケーションの中で準備するかことから外部のデータセットとして準備することとした。

Pytorchの開発スピードは早く、またベンチマークのためのライブラリオブジェクトとスクリプトは仮想環境での実行形式がバージョンの更新毎に提供されている。実際のハードウェアでの性能評価では安定したバージョンを設定することが必要となった。富岳NEXTと類似の命令セット構成であるGrace Hopper向けと現在の富岳向けにPytorchのバージョンの選定を行った。

1.8.7 来年度（詳細設計フェーズ）に向けた重要項目

PETScを複数GPUが相互に直接接続されている環境で実行するためにはPETScSF通信ライブラリを利用する必要がある。これは開発版であるためPETSc開発コミュニティとの連携を強化する必要がある。Fortranインターフェースが利用できると国内でのアプリケーションからの利用者を拡大することができるため、PETSc ライブラリの構築のために必要なコンパイラの特有の機能の整理を行う。GPUでの疎行列ベクトル演算のSpMVはGPUベンダが提供しているライブラリを利用しているが、Kokkos KernelsによるSpMVのバックエンドも実装されている。CPUとGPUが混在した環境でのデータ分散をPETScが担うかKokkosの機能を活用するかを評価する必要がある。

LLMの学習においては通信がボトルネックとなることがある。高度な学習のためには大規模なパラメータの利用が必須であるため、複数GPUが相互に直接接続されている環境での実行が効率的におこなわれるか検討が必要である。